ESCUELA SUPERIOR DE INGENIERÍA

Programación en Internet

Grado en Ingeniería Informática

Creación de un sitio web dinámico utilizando Bootstrap, AngularJS y jQuery

Pablo Antonio García Chacón

Supervisores:

Juan Boubeta Puig y Guadalupe Ortiz Bellot

Cádiz, 4 de enero de 2016

Contenido

Introducción

Este tutorial describe de forma detallada la creación de un sitio web con elementos estáticos, así como dinámicos, haciendo uso de HTML5 y CSS. El sitio hará invocaciones al servicio web mediante JavaScript para la parte dinámica del mismo.

Sitio web

1. Prototipo del sitio web

El sitio web se compondrá de cinco páginas distintas:

- Página principal o de bienvenida
- Proyectos
- Profesores
- Contacto
- Administración

El prototipo del sitio web se trata de un diseño esquemático de la estructura y del diseño y disposición de los elementos que tendrá de una forma sencilla. La herramienta usada para realizar el prototipo es Pencil.

1.1. Página principal

La página principal tendrá una sección superior con el menú del sitio, que será común a todas las páginas, además del título del proyecto, el nombre de la universidad y el logo del sitio. Tendrá una imagen de fondo.

A continuación tendrá una sección principal y central de bienvenida con una descripción del sitio, enlaces a las distintas páginas y un mapa de localización.

Por último, habrá una sección lateral derecha con un resumen de noticias destacadas, que serán los últimos seis proyectos publicados (dinámicamente obtenidos), y abajo un resumen de fechas destacadas.

1.2. Proyectos

La página de proyectos también tendrá una sección superior con una imagen de fondo, el menú común a todas las páginas y el título de la página.

La sección principal tendrá una lista en forma de distintos bloques de todos los proyectos publicados, mostrando toda la información disponible de los mismos. Cada proyecto incluirá una imagen estática.

Entre la sección superior de presentación y la principal con la información, habrá un pequeño bloque decorativo con el título del objetivo principal de la página: todos los proyectos.

Por último, habrá una sección lateral a la derecha que contendrá estadísticas de los proyectos. Se mostrará en un gráfico circular generado gracias a una librería externa JavaScript, la representación de las categorías empleadas en los distintos proyectos.

1.3. Profesores

La página de profesores tendrá un diseño similar a las anteriores en cuanto a sección posterior y también incluirá el bloque central que presenta el tema principal de la página.

La siguiente sección ocupará todo el ancho del contenedor y mostrará una descripción de los profesores y a continuación una tabla dinámica con información de los mismos.

La siguiente sección sería la de perfiles, donde se mostrará a modo de carta un avatar estático del profesor junto con su nombre, para cada uno de los profesores.

Por último, habrá una sección lateral derecha con estadísticas de la participación de los profesores en los proyectos en desempeño de su labor como tutores o cotutores. Incluirá además un gráfico circular haciendo uso de la misma librería JavaScript anteriormente mencionada.

1.4. Contacto

La página de contacto tendrá una sección superior con el menú y el título de la página al igual que las páginas anteriores.

La sección principal contendrá un formulario de contacto con los administradores de la página.

Habrá una sección lateral a la derecha con un mapa donde se indica la ubicación de la universidad.

1.5. Administración

La página de administración tendrá una sección superior con el menú y el título de la página al igual que las páginas anteriores. También incluirá el bloque que describe la página.

La sección principal de esta página mostrará cuatro botones circulares para realizar cada una de las acciones *CRUD* que nos permite el servicio web. Al pulsar sobre uno de estos botones se mostrará el formulario pertinente.

2. Recursos utilizados

Además de utilizar las tecnologías web HTML5, CSS3 y JavaScript se emplearán en este proyecto una serie de *frameworks* y librerías que nos permitan conseguir más en menos tiempo.

2.1. Bootstrap

Se ha elegido Twitter Bootstrap como *framework* CSS fundamentalmente por su sistema de *grid* o cuadrículas que permite organizar la información de la web por columnas de igual tamaño. En concreto, utiliza un sistema de doce columnas.

Además, proporciona una serie de estilos básicos para casi cualquier elemento que se pueda imaginar que proporcionan una buena base y punto de partida para comenzar a dar estilos más personalizados al sitio.

2.2. AngularJS

Se ha elegido AngularJS como *framework* JavaScript para la interacción con el servicio web, la manipulación del DOM y la validación.

La principal razón por la que se ha elegido AngularJS es porque separa la lógica de la aplicación JavaScript de la manipulación del DOM.

2.3. *jQuery*

Se utilizará la extendidísima librería jQuery para referenciar los elementos del DOM y hacer cambios en el mismo desde nuestro fichero principal JavaScript.

3. Desarrollo del sitio web

En esta sección se explicará el desarrollo del sitio web con todas sus páginas. Para cada una de ellas, se detallarán las razones del uso de las distintas etiquetas HTML, especialmente en los elementos del diseño más significativos; también se mostrará el CSS usado para dar estilo a la página y el código JavaScript utilizado, principalmente en relación a la interacción con el servicio web.

3.1. Inicio

3.1.1. Estructura

Abrimos el fichero *index.html* y añadimos el código inicial de plantilla HTML5:

```
<!DOCTYPE html>
<html lang="es">
<head>
        <meta charset="UTF-8">
</head>
<body>

</body>
</html>
```

Añadimos etiquetas *meta* con información sobre la página como el título, la descripción, la codificación de caracteres que será UTF-8 y el icono del sitio:

```
<meta name="description" content="Página de inicio y bienvenida del sitio web
Gestión Proyectos">
<meta charset="UTF-8">
<title>Proyectos - Inicio</title>
<link rel="icon" href="img/logo.ico">
```

Si recordamos el prototipo de la página principal tiene una sección superior, empecemos añadiendo esa sección al principio del cuerpo (<*body*>) y para ello utilizaremos la etiqueta HTML5 <*section*>, que sirve para añadir un bloque (como el clásico <*div*>) pero con la semántica de **sección**:

```
<section id="upper-section">
```

```
</section>
```

Primero tendremos el menú de navegación, el cual incluiremos dentro de un * * padre para fijar un ancho y usaremos una etiqueta HTML5 *<nav>* que se trata de un bloque con la semántica de **navegación**.

La estructura del menú está preparada para responder a los estilos que Bootstrap aplica. El contenido del *nav* está envuelto por un *div* contenedor y dentro del mismo habrá una cabecera con el logo del sitio web, que enlazará con la página web del departamento haciendo uso de la etiqueta *<a>* y, a continuación, el cuerpo del menú, que contendrá una lista desordenada (*<ul>)* con ítems (*<li>*) que contienen enlaces a las distintas páginas del sitio:

```
<div class="container mini-container">
        <nav class="navbar navbar-default navbar-static-top">
                <div class="container-fluid">
                        <div class="navbar-header">
                                <button type="button" class="navbar-toggle
collapsed" data-toggle="collapse" data-target="#navbar">
                                        <span class="sr-only">Cambiar
navegación</span>
                                        <span class="icon-bar"></span>
                                        <span class="icon-bar"></span>
                                        <span class="icon-bar"></span>
                                </button>
                                <a href="http://departamentos.uca.es/C137/"
target="_blank" class="navbar-brand">
                                        <img src="img/logo_white.png"
alt="Logo">
                                </a>
                        </div>
                        <div id="navbar" class="navbar-collapse collapse
navbar-right">
                                <ul class="nav navbar-nav">
                                        <li class="active">
                                                <a
href="index.html">Inicio</a>
                                        </li>
                                        <li>
                                                <a
href="tfg.html">Proyectos</a>
                                        </li>
                                        <li>
```

```
                                                <a
href="profesores.html">Profesores</a>
                                            </li>
                                            <li>
                                                <a
href="contactar.html">Contacto</a>
                                            </li>
                                            <li>
                                                <a
href="index.html#map">Mapa</a>
                                            </li>
                                            <li>
                                                <a
href="administracion.html">Administración</a>
                                            </li>
                                    </ul>
                            </div>
                    </div>
            </nav>
</div>
```

Por último, en la sección superior tenemos el título del proyecto que usa una etiqueta de cabecera principal *<h1>*, el nombre de la Universidad de Cádiz como subtítulo que usa una cabecera de menor tamaño *<h2>* y el logo, una imagen *<img>*:

```
<div class="col-md-12">
        <section id="logo-section">
                <div class="row">
                        <div class="logo-container row">
                                <div class="container">
                                        <h1>Gestión de proyectos</h1>
                                        <h2>Universidad de Cádiz</h2>
                                </div>
                                <img src="img/logo_white.png" alt="Logo">
                        </div>
                </div>
        </section>
</div>
```

Para los logos, tanto del menú como de la sección superior, hemos utilizado la etiqueta *<img>* para añadir una imagen. La fuente de la imagen se indica con el atributo *src* al que se le da el valor de la ubicación relativa del fichero. El atributo *alt* se utiliza

para describir la imagen mostrada con el objetivo de conseguir que el sitio sea **accesible**, por ejemplo, que pueda ser leído por un lector de páginas web usado por alguna persona con discapacidad visual.

Después de la sección superior tenemos la sección principal, para la que también usamos la etiqueta <*section*> y tiene un único hijo inmediato: un *div* contenedor.

```
<section id="main-section">
        <div class="container">
        </div>
</section>
```

Esta sección tiene, a su vez, dos secciones: la sección principal de bienvenida con información diversa del sitio, y una sección lateral con información secundaria.

Primero, tenemos un bloque de bienvenida:

```
<div class="row">
      <div class="col-md-12">
            <h2>¡Bienvenido!</h2>
            <p>Explora nuestro sitio web para la gestión e información de
Proyectos de Fin de Grado. Consulta toda la información que necesites sobre
los proyectos, así como de los estudiantes y los profesores. Accede a nuestro
panel de administrador para añadir o actualizar la información a lo más
reciente. No dudes en contactar con nosotros.</p>
            <p>¡Disfruta!</p>
      </div>
</div>
```

A continuación, una serie de enlaces a los distintos sitios de interés del sitio en forma de bloques coloridos:

```
<div class="row">
        <a href="tfg.html">
                <div id="projects-block" class="col-md-3 iconic-information-
block">
                        <span class="glyphicon glyphicon-bookmark"></span>
                        <p>Proyectos</p>
                </div>
        </a>
        <a href="profesores.html">
                <div id="professors-block" class="col-md-3 iconic-
information-block">
                        <span class="glyphicon glyphicon-user"></span>
                        <p>Profesores</p>
```

```
                </div>
        </a>
        <a href="administracion.html">
                <div id="admin-block" class="col-md-3 iconic-information-
block">
                        <span class="glyphicon glyphicon-dashboard"></span>
                        <p>Administración</p>
                </div>
        </a>
</div>
```

Por último, un bloque que contiene el mapa de Google Maps centrado en la ubicación de la ESI.

```
<div class="row">
        <div class="col-md-12">
                <h3>¿Dónde nos puedes localizar?</h3>
                <address>Nos encontramos en la Escuela Superior de
Ingeniería de la Universidad de Cádiz. Avenida Universidad de Cadiz, 10,
11519 Puerto Real, Cádiz.</address>
                <div id="map"></div>
        </div>
</div>
```

La sección secundaria dentro de la sección principal se trata de una sección lateral para la que se usa una etiqueta HTML5 *<aside>* que al igual que las vistas anteriormente, se trata de un bloque, esta vez con la semántica de sección lateral. Se usan dos etiquetas *aside* ya que tenemos dos piezas de información distintas: las noticias destacadas y las fechas importantes:

```
<section id="aside" class="col-md-4">
        <aside id="news">
                <h3 class="aside-header">Noticias destacadas</h3>
                <ul class="list-unstyled">
                        <li ng-repeat="project in projects"><strong>{{
project.title }}</strong>, por {{ project.studentName }}</li>
                </ul>
        </aside>
        <aside id="dates">
                <h3 class="aside-header">Fechas importantes</h3>
                <ul class="list-unstyled">
```

```
                        <li><strong>III Taller de Drupal</strong>,
3/12/2015</li>
                        <li><strong>V Conferencia de Programación Orientada
a Objetos</strong>, 16/11/2015</li>
                        <li><strong>XI Hackathon Cádiz</strong>,
9/11/2015</li>
                </ul>
        </aside>
</section>
```

La última parte de la página principal es el pie de página, común a todas las demás páginas del sitio web. El pie de página contiene información sobre la autoría de la web, contacto con el autor mediante el valor *mailto* de la etiqueta <*a*> y, finalmente, el logo.

```
<footer>
        <div class="col-md-12">
                <p>&copy; 2015 Pablo Antonio García Chacón</p>
                <a
href="mailto:pabloantonio.garciachacon@alum.uca.es">pabloantonio.garciachacon
@alum.uca.es</a>
        </div>
        <div class="col-md-12">
                <img src="img/logo.png" alt="Logo">
        </div>
</footer>
```

3.1.2. Estilo

Como se ha podido ver, se han usado clases CSS en las etiquetas HTML de la página principal. Estas clases se utilizarán en los correspondientes ficheros CSS importados por la página para aplicar estilos al HTML.

La página de inicio cargará dos ficheros CSS: *app.css* que será común a todas las páginas, y *index.css* que contiene estilos usados únicamente en esta página. También hay que recordar importar el fichero CSS de Bootstrap.

Para cargar los ficheros de estilo hay que usar la etiqueta <*link*> y los añadimos bajo la sección <*head*> de nuestro fichero HTML:

```
<link rel="stylesheet" type="text/css" href="css/bootstrap.css">
<link rel="stylesheet" href="css/app.css">
```

```
<link rel="stylesheet" href="css/index.css">
```

Antes de explicar los estilos aplicados a esta página veremos una serie de estilos propios de Bootstrap utilizado en todo nuestro sitio. Los estilos que vamos a ver sirven para estructurar la organización en bloques o columnas y nos ayudan a posicionar los distintos elementos del HTML de una forma más sencilla y organizada:

La clase ***container*** la utilizamos siempre como primer hijo de cada sección superior y principal. Esta clase aplica un ancho fijo al bloque dependiendo de la resolución de la pantalla (usando *media queries*), además de unos márgenes laterales automáticos y *padding* lateral de 15px:

```
<section id="upper-section">
        <div class="container mini-container">
        </div>
</section>

<section id="main-section">
        <div class="container small-container">
        </div>
</section>
```

La clase ***row*** la utilizamos para añadir una fila. Se utiliza como contenedor de una estructura de distinta complejidad de elementos HTML que se quieran organizar en un bloque horizontal (fila) que ocupe todo el ancho (a diferencia del contenedor, no tiene anchura fija). Además tiene unos márgenes laterales de -15px. Esta clase se utiliza comúnmente en todas las páginas del sitio web.

```
<div class="row">
```

Las clases ***col-md-**** y, en menor medida, ***col-sm-**** son las clases que definen un *grid* o cuadrícula. Bootstrap incorpora un sistema de cuadrículas de 12 columnas. Aplicando las clases *col-md-** donde * puede variar de 1 a 12, podemos indicar qué ancho queremos que aplique el bloque donde se utilice esa clase. La diferencia entre *col-md* y *col-sm* es que la primera se aplica para pantallas con una resolución mínima de 992px y *col-sm* para 768px.

Así, por ejemplo, si definimos lo siguiente

```
<div class="col-md-4"></div>
<div class="col-md-4"></div>
```

```
<div class="col-md-4"></div>
```

tendremos tres bloques del mismo ancho dispuestos horizontalmente uno junto a otro.

Estas clases se utilizan por todo el sitio para estructurar la información, por ejemplo, para indicar que la sección primaria de la sección principal ocupe 8 columnas y la sección lateral 4 columnas a la derecha de la anterior:

```
<section id="main-section">
        <div class="container">
                <section id="welcome" class="col-md-8">
                </section>
                <section id="aside" class="col-md-4">
                </section>
        </div>
</section>
```

Estos estilos para estructurar los bloques es una de las principales razones por las que se ha elegido Bootstrap como *framework* CSS a utilizar en nuestro sitio.

Ahora, pasemos a los estilos de esta página:

Primero, en nuestro fichero *app.css* aplicamos estilos generales al cuerpo. Importamos una fuente propia ubicada en la carpeta *fonts* declarando una *@font-face* y aplicando la propiedad *font-family* a *body* cambiamos la fuente de todo el documento:

```
@font-face {
  font-family: 'Montserrat';
  font-style: normal;
  font-weight: 400;
  src: local('Montserrat-Regular'), url(../fonts/montserrat.woff2)
format('woff2');
  unicode-range: U+0000-00FF, U+0131, U+0152-0153, U+02C6, U+02DA, U+02DC,
U+2000-206F, U+2074, U+20AC, U+2212, U+2215, U+E0FF, U+EFFD, U+F000;
}

html {
        height: 100%;
}

body {
```

```
        font-family: 'Montserrat', Calibri, sans-serif;
        padding-bottom: 20px;
        background-color: #fff;
}
```

A continuación aplicamos estilos a los distintos elementos del menú superior para conseguir un bloque de navegación con un fondo transparente, letras de color blanco y un borde inferior. Además de reajustar los márgenes (propiedades *margin*, *margin-top*, *margin-left*, etc.) y rellenos (propiedades *padding*, *padding-right*, *padding-bottom*, etc.):

```
.navbar-brand {
        padding: 10px;
}

.navbar-brand img {
        height: 100%;
}

#upper-section nav {
        margin-top: 10px;
        background-color: transparent;
        border-bottom: 2px solid #fff;
}

#upper-section nav a {
        color: #fff;
        background-color: transparent;
}

#upper-section nav a:hover {
        text-decoration: underline;
}

#upper-section button {
        border-color: #fff;
}

#upper-section button:hover, #upper-section button:focus {
        background-color: rgba(255, 255, 255, 0.3);
}

#upper-section button span {
```

```
        background-color: #fff;
}
```

También hemos añadido contenedores propios con un ancho determinado y relativo al ancho total del padre:

```
.mini-container {
        width: 55%;
}

.small-container {
        width: 70%;
}
```

Estilos generales para los bloques de información (usados en esta página como enlaces a distintas páginas del sitio) y también hemos ajustado las cabeceras de las secciones laterales modificado el *margin* y el *padding*, así como añadiendo un borde inferior que haga de separación con el contenido de debajo:

```
.actions-header-container h2 {
        font-size: 36px;
}

.aside-header {
        margin-top: 0px;
        border-bottom: 2px solid #aaa;
        text-align: center;
        padding-bottom: 10px;
}
```

Por último, en el fichero CSS común *app.css* se definen los estilos del pie de página, añadiendo márgenes y *padding*, centrando el texto y mostrando un borde superior de separación:

```
footer {
        border-top: 2px solid #999;
        width: 90%;
        height: 50px;
        margin: 0 auto;
        margin-top: 15px;
        padding: 15px;
        text-align: center;
```

```
        color: #999;
}

footer p {
        margin-bottom: 0px;
}

footer a {
        text-decoration: none;
        color: #999;
        font-size: 12px;
}

footer a:hover, a:focus {
        color: #999;
}

footer img {
        height: 60px;
        margin-top: 20px;
        margin-bottom: 20px;
}
```

Además del fichero *app.css* se cargan estilos propios de esta página con el fichero *index.css.*

Primero, establecemos un ancho fijo en píxeles de la sección superior y una imagen de fondo. Añadir una imagen de fondo con la propiedad *background-size: cover* tiene la ventaja sobre incluir una etiqueta <*img*> dentro de un bloque de que las dimensiones del fondo cambian si redimensionamos la ventana del navegador.

```
#upper-section {
        height: 600px;
        background-image: url(../img/background.jpg);
        background-size: cover;
}
```

El siguiente estilo lo aplicamos al título, subtítulo y logo que aparecen en la sección posterior. Aparecerán centrados, se aplicará un tamaño fijo a la imagen del logo y se usará la propiedad *text-transform: uppercase* para transformar a mayúsculas todo el texto contenido en ese bloque:

```
.logo-container {
        width: 100%;
        text-align: center;
        text-transform: uppercase;
}

.logo-container h1 {
        font-size: 48px;
        color: #fff;
}

.logo-container h2 {
        font-size: 16px;
        color: #fff;
        margin-top: 0px;
}

.logo-container img {
        height: 75px;
}
```

Comenzamos los estilos de la sección principal añadiendo algo de separación de la sección superior:

```
#main-section {
        margin-top: 40px;
}
```

La sección de bienvenida tiene un color de fondo y un borde redondeado en las esquinas usando la propiedad *border-radius*. También se estilan las cabeceras y se elimina el subrayado de los enlaces con *text-decoration: none*:

```
#welcome {
        background-color: #eee;
        border-radius: 5px;
        padding: 35px;
        margin-bottom: 40px;
}

#welcome h2 {
        font-size: 36px;
        margin-top: 0px;
        margin-bottom: 15px;
```

```
}

#welcome a {
        text-decoration: none;
}

.iconic-information-block {
        margin: 0px 5px 25px 5px;
        color: #eee;
}
```

En nuestro sitio web hemos optado por relacionar cada una de las secciones principales con una serie de matices de color:

- Proyectos: naranja
- Profesores: morado
- Administración: verde

En cada una de estas páginas la gama de colores utilizados serán los mencionados. Así, en la página principal se introduce al usuario a esta correspondencia de colores utilizando cada uno de ellos en los enlaces con forma de bloque coloridos:

```
#projects-block {
        background-color: #fba768;
}

#professors-block {
        background-color: #7367ae;
}

#admin-block {
        background-color: #4cb547;
}
```

Cada uno de estos bloques además de un texto que indica la sección a la que se llegará cuando se haga clic sobre ellos, un icono. Los iconos son proporcionados por Bootstrap y, como se puede ver en el HTML anteriormente mostrado, se usan así:

```
<span class="glyphicon glyphicon-bookmark"></span>
```

En este caso, se mostraría el icono de un marcapáginas. La página oficial de Bootstrap ofrece un listado de las clases que usar en lugar de *glyphicon-bookmark* para usar otro icono distinto.

La siguiente propiedad establece una altura fija al mapa de Google:

```
#map {
        height: 400px;
}
```

Por último, añadimos los estilos de la sección lateral añadiendo un borde inferior a cada uno de los ítems de la lista excepto al último. Para aplicar estilos al último elemento usamos la pseudoclase *last-child*:

```
#news, #dates {
        padding-left: 40px;
}

#news li, #dates li {
        border-bottom: 1px solid #eee;
        padding: 10px 0px;
}

#news li:last-child, #dates li:last-child {
        border-bottom: none;
}

#dates {
        margin-top: 40px;
}
```

Tras aplicar todos los estilos, la página principal se ve como en la Figura 1:

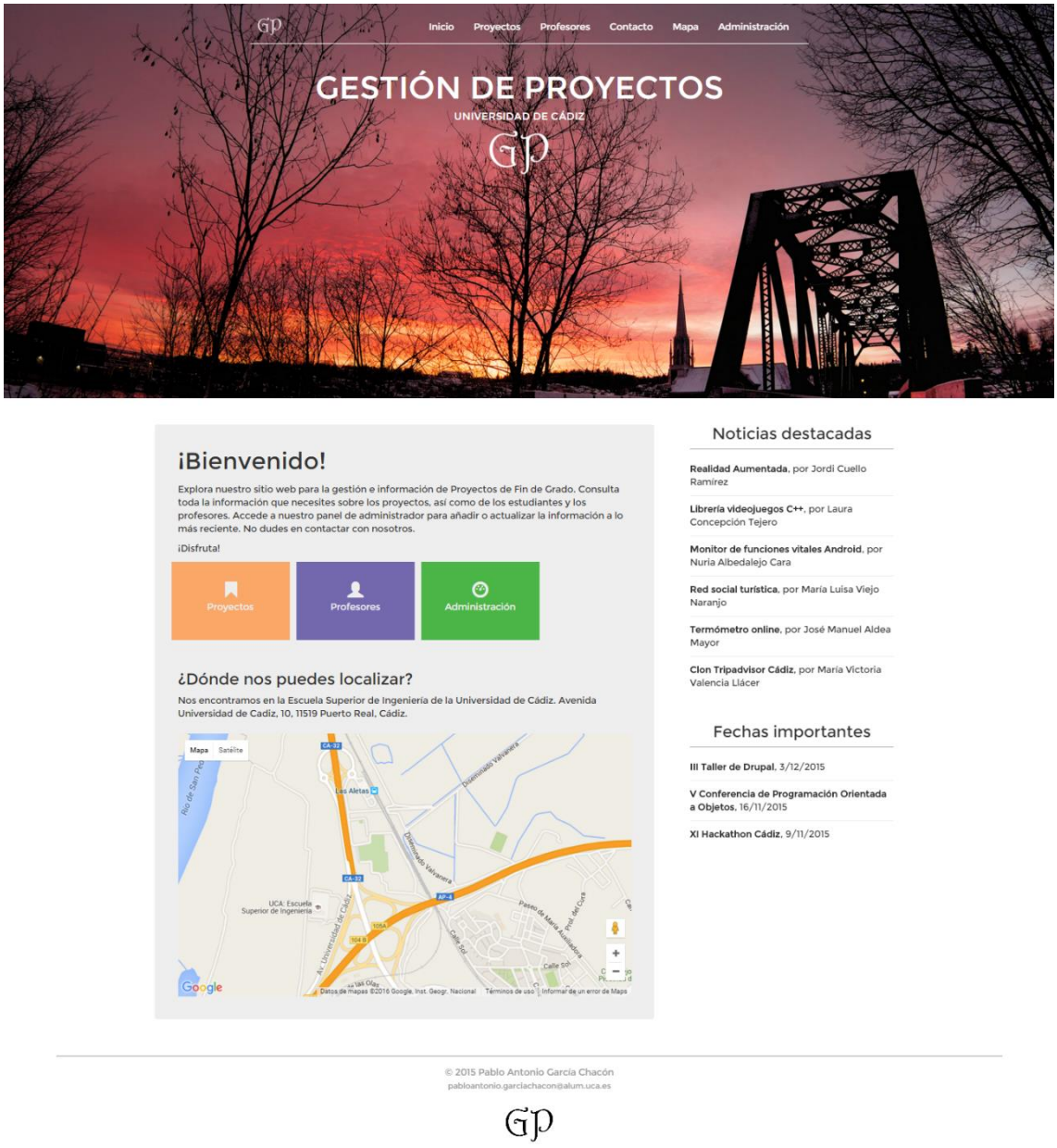

Figura 1. Página principal

3.1.3. JavaScript

En la página principal cargamos el mapa de Google haciendo uso de su API para JavaScript. Primero, añadimos el fichero fuente de Google Maps bajo la sección *<head>*:

```
<script src="https://maps.googleapis.com/maps/api/js"></script>
```

En la carpeta *js* creamos el fichero *map.js* al que añadimos:

```
function initialize() {
        var mapCanvas = document.getElementById('map');
    var mapOptions = {
      center: new google.maps.LatLng(36.53949, -6.20214),
      zoom: 15,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    }
    var map = new google.maps.Map(mapCanvas, mapOptions);
}

google.maps.event.addDomListener(window, 'load', initialize);
```

Con JavaScript se obtiene el elemento del DOM con la id “map”, que se trataría de un *div* con un alto fijado por CSS. En el atributo *center* indicamos las coordenadas donde queremos que aparezca centrada la vista sobre el mapa.

Finalmente, cargamos el recurso a continuación del fichero fuente de Google Maps:

```
<script src="js/map.js"></script>
```

3.1.3.1. AngularJS

En esta página también hacemos uso de AngularJS para cargar los últimos proyectos publicados en la barra lateral.

Para usar AngularJS lo primero que hacemos es cargar el fichero fuente:

```
<script src="js/angular.js"></script>
```

También creamos un fichero llamado *app.js* dentro del directorio *js* y será nuestro fichero principal de Angular.

AngularJS funciona por directivas, es decir, atributos dentro de una etiqueta HTML que definen un comportamiento. Para indicar que nuestra página está usando AngularJS se usa la directiva *ng-app*, como observamos en la etiqueta *html:*

```
<html lang="es" ng-app="GestionProyectos">
```

El nombre *GestionProyectos* permitirá referenciar la aplicación Angular desde nuestro fichero JavaScript principal.

A continuación hay que definir un controlador sobre el ámbito que queremos que se apliquen las funciones de Angular. Normalmente, se declara en el *body:*

```
<body ng-controller="IndexCtrl">
```

Si navegamos hacia la lista de noticias importante, donde se muestran los últimos proyectos, podemos ver:

```
<li ng-repeat="project in projects"><strong>{{ project.title }}</strong>, por
{{ project.studentName }}</li>
```

La directiva *ng-repeat* es una directiva de Angular que itera sobre la variable *projects* y extrae cada elemento de la iteración en otra variable temporal *project.* Ésta se utiliza para acceder a sus propiedades mediante el operador . y se muestra su resultado mediante el uso de doble llave *{{ }}*.

La variable *projects* se define sobre el objeto de Angular *$scope,* que sirve para comunicarse con el DOM.

Para darle la funcionalidad requerida a nuestra aplicación definimos una aplicación Angular y un controlador en nuestro fichero *app.js:*

```
var app = angular.module('GestionProyectos', []);
```

```
app.controller('IndexCtrl', function($scope, ws) {

});
```

En este controlador invocaremos un método de un servicio que a continuación explicaremos y cuando su ejecución termine (*then*) se creará la variable *projects* con la información de los últimos seis proyectos. Esta variable se asignará a la variable *projects* del objeto *$scope* anteriormente mencionado, lo que la hará accesible desde el HTML:

```
ws.getAllStudents().then(function (data) {
        var count = 0;
        var limit = 6;
        var projects = [];
        $.each(data, function(key, student) {
                projects.push({
                        "studentName": (student.nombre + " " +
student.primerApellido + " " + student.segundoApellido).trim(),
                        "title": student.tituloProyecto
                });

                count++;

                return count < limit;
        });

        $scope.projects = projects;
});
```

Para las invocaciones al servicio web necesitamos un servicio que definimos al final del fichero *app.js*:

```
app.factory('ws', function($http, $q) {
        var host = "http://localhost";
        var port = "8080";
        var name = "GestionProyectos";
        var service_path = "api/estudiantes";
        var full_path = host + ":" + port + "/" + name + "/" + service_path;

        var get_all_estudiantes_path = "/";
        var get_estudiante_path = "/buscar";
        var create_estudiante_path = "/crear";
```

```
        var update_estudiante_path = "/actualizar";
        var delete_estudiante_path = "/eliminar";

        return {
                getAllStudents: function() {
                        var url = full_path + get_all_estudiantes_path;
                        return $http.get(url).then(function(response) {
                                return response.data;
                        });
                }
        }
});
```

El servicio (*factory*) devuelve un objeto con una serie de métodos que llevan a cabo las distintas invocaciones al servicio.

De momento, incluye el atributo *getAllStudents*, al que se le asigna una función que, al llamarla, hace una petición GET y devuelve el resultado obtenido de la invocación del método del servidor correspondiente a la obtención de un JSON con todos los proyectos.

3.2. Proyectos

3.2.1. Estructura

Del mismo modo que hicimos anteriormente, añadimos la estructura básica HTML5 y etiquetas *meta*:

```
<!DOCTYPE html>
<html lang="es">
<head>
        <meta name="description" content="Información de los Proyectos de
Fin de Grado">
        <meta charset="UTF-8">
        <title>Proyectos - Proyectos</title>
        <link rel="icon" href="img/logo.ico">
</head>
</html>
```

La estructura inicial es idéntica a la anterior página y así será en todas las demás: tenemos una sección superior y su primer hijo es el menú. Basta con copiar lo realizado en el punto anterior. En este caso, no obstante, en la última parte de la sección superior en lugar de mostrar un título, subtítulo y logo solamente mostramos el título de la página:

```
<div id="title" class="col-md-12">
        <h1>Proyectos de fin de grado</h1>
</div>
```

La estructura básica de la sección principal es similar a las demás: una sección con un contenedor de anchura fija como hijo. Lo primero que añadimos es el bloque de información con el título del elemento principal de la página. Nos referiremos a este bloque como pestaña informativa, ya que es un elemento usado en diversos lugares de nuestro sitio:

```
<div class="row">
        <div class="col-md-3 iconic-information-block">
                <span class="glyphicon glyphicon-bookmark"></span>
                <p>Todos los proyectos</p>
        </div>
</div>
```

El siguiente hijo de la sección principal es la sección de los proyectos. Primero, contiene una caja de búsqueda para filtrar por nombre de título. Usamos la etiqueta *<input type=”text”>* para representar un campo de entrada de texto:

```
<div id="filter-container">
        <div class="input-group">
                <span class="input-group-addon">
                        <span class="glyphicon glyphicon-search"></span>
                </span>
                <input type="text" class="form-control" ng-model="title"
placeholder="Filtrar por título">
        </div>
</div>
```

A continuación del campo de búsqueda habrá una sucesión de paneles. Cada uno de estos paneles contendrá la información de un proyecto. Primero, una cabecera con el título del proyecto, después un bloque con el cuerpo de la información usando la etiqueta *<hr>* en mitad para dividir la información y, finalmente, un pie con la categoría del proyecto. Las etiquetas usadas se tratan de * * porque tratamos con bloques genéricos, sin necesidad de una semántica adicional que proporciona HTML5:

```
<div class="panel panel-default" ng-repeat="project in projects |
filter:{title: title}">
        <div class="panel-heading">{{ project.title }}</div>
        <div class="panel-body">
                <div class="col-md-8">
                        <p><strong>Autor: </strong>{{ project.studentName
}}</p>
                        <p><strong>Tutores: </strong>{{ project.tutor }}
<span ng-hide="!project.cotutor || project.cotutor.length === 0">y {{
project.cotutor }}</span></p>
                        <hr>
                        <p><strong>Estado del proyecto: </strong>{{
project.state }}</p>
                        <p ng-hide="!project.date || project.date.length ===
0"><strong>Fecha de presentación: </strong>{{ project.date }}</p>
                        <p ng-hide="!project.mark || project.mark.length ===
0"><strong>Calificación: </strong>{{ project.mark }}</p>
                </div>
                <div class="project-image-container col-md-4">
                        <img src="img/{{ project.img }}" alt="{{
project.imgDescription }}" class="img-circle">
                </div>
```

```
        </div>
        <div class="panel-footer">
                {{ project.topic }}
        </div>
</div>
```

El último elemento de la sección principal se trata de la barra lateral con las estadísticas de las categorías de los proyectos. La estructura es idéntica a las barras laterales de la página de inicio, y así será en el resto de páginas donde haya una barra lateral:

```
<aside id="stats" class="col-md-4">
        <h3 class="aside-header">Estadísticas</h3>
        <ul class="list-unstyled">
                <li>
                        <strong>Total de proyectos: </strong>{{
homeAutomation + health + tourism }}
                </li>
                <hr>
                <li>
                        <span id="home-icon" class="glyphicon glyphicon-
home"></span>
                        <span><strong>Proyectos de domótica: </strong><span
id="homeAutomation-count">{{ homeAutomation }}</span></span>
                </li>
                <li>
                        <span id="health-icon" class="glyphicon glyphicon-
heart"></span>
                        <span><strong>Proyectos de salud: </strong><span
id="health-count">{{ health }}</span></span>
                </li>
                <li>
                        <span id="tourism-icon" class="glyphicon glyphicon-
sunglasses"></span>
                        <span><strong>Proyectos de turismo: </strong><span
id="tourism-count">{{ tourism }}</span></span>
                </li>
        </ul>
        <div class="col-md-12">
                <canvas id="chart" width="150" height="150"></canvas>
        </div>
</aside>
```

En esta sección lateral hacemos uso de la etiqueta *<canvas>* donde se mostrará el gráfico circular. Es un elemento de HTML5 que sirve para dibujar gráficos en la web a través de scripting.

Al final del *body* tenemos el *footer*, que es el mismo en todas las páginas.

3.2.2. Estilo

Los estilos usados en la página de los proyectos son el fichero común *app.css* y los estilos propios de esta página, *tfg.css*.

Primero, en la sección superior, establecemos una imagen de fondo y estilamos el título principal:

```
#upper-section {
        height: 300px;
        background-image: url(../img/background-tfg.jpg);
        background-size: cover;
}

#title {
        text-transform: uppercase;
        text-align: center;
        margin-top: 40px;
}

#title h1 {
        font-size: 40px;
        color: #fff;
}
```

Damos el estilo anaranjado correspondiente a la sección de proyectos a la pestaña informativa:

```
.iconic-information-block {
        margin: 0 auto;
        float: none;
        background-color: #fb914e;
        color: #fff;
        text-transform: uppercase;
}
```

Añadimos margen a los lados del contenedor de los proyectos. Separamos el campo de búsqueda con un margen inferior y establecemos su ancho a la mitad del contenedor padre y lo centramos con *margin: 0 auto*:

```
#projects-section {
        margin: 30px auto;
}

#projects-section #filter-container {
        margin-bottom: 20px;
}

#projects-section #filter-container .input-group {
        width: 50%;
        margin: 0 auto;
}
```

A continuación añadimos los estilos al panel con la información de un proyecto. Un borde redondeado y de un color naranja, una cabecera con un fondo naranja ligeramente distinto y un pie con un fondo naranja más claro.

También aplicamos un ancho y alto determinados a la imagen para que cambie conforme cambie el tamaño de la ventana en relación al tamaño del padre, y la posicionamos verticalmente aplicando márgenes superior e inferior relativos. Establecemos un ancho máximo con la propiedad *max-width* para evitar que se muestren imágenes demasiado grandes.

```
#projects-section .panel-default {
        border-color: #fbcb9e;
}

#projects-section .panel-heading {
        font-weight: bold;
        background-color: #fba768;
        border-color: #fbcb9e;
        font-size: 20px;
}

#projects-section hr {
        border-color: #fbcb9e;
}

.project-image-container {
```

```
        text-align: center;
}

#projects-section img {
        height: 70%;
        width: 70%;
        margin-top: 25%;
        margin-bottom: 25%;
        max-width: 200px;
}

#projects-section .panel-footer {
        border-color: #fbcb9e;
        text-align: center;
        background-color: #fbcb9e;
        font-weight: bold;
}
```

Podemos ver en la estructura HTML que usamos las clases *panel* y derivados (*panel-default, panel-heading*…). Esta se trata una clase de Bootstrap para paneles. Un panel es como una ficha o una tarjeta: un bloque con borde alrededor, contenido, cabecera y pie.

Finalmente, aplicamos estilos triviales a la sección lateral con las estadísticas. Destacamos el uso de la pseudoclase *first:child* donde aplicamos un estilo al primer ítem de la lista para darle apariencia de elemento resaltado:

```
#stats {
        margin: 30px auto;
        padding-left: 40px;
}

#stats hr {
        border-color: #ddd;
}

#stats li {
        padding: 5px 0px;
}

#stats li:first-child {
        font-size: 16px;
}
```

```
#home-icon {
        color: #46bfbd;
}

#health-icon {
        color: #f7464a;
}

#tourism-icon {
        color: #fed750;
}
```

Al aplicar estos estilos la página se puede ver en la Figura 2:

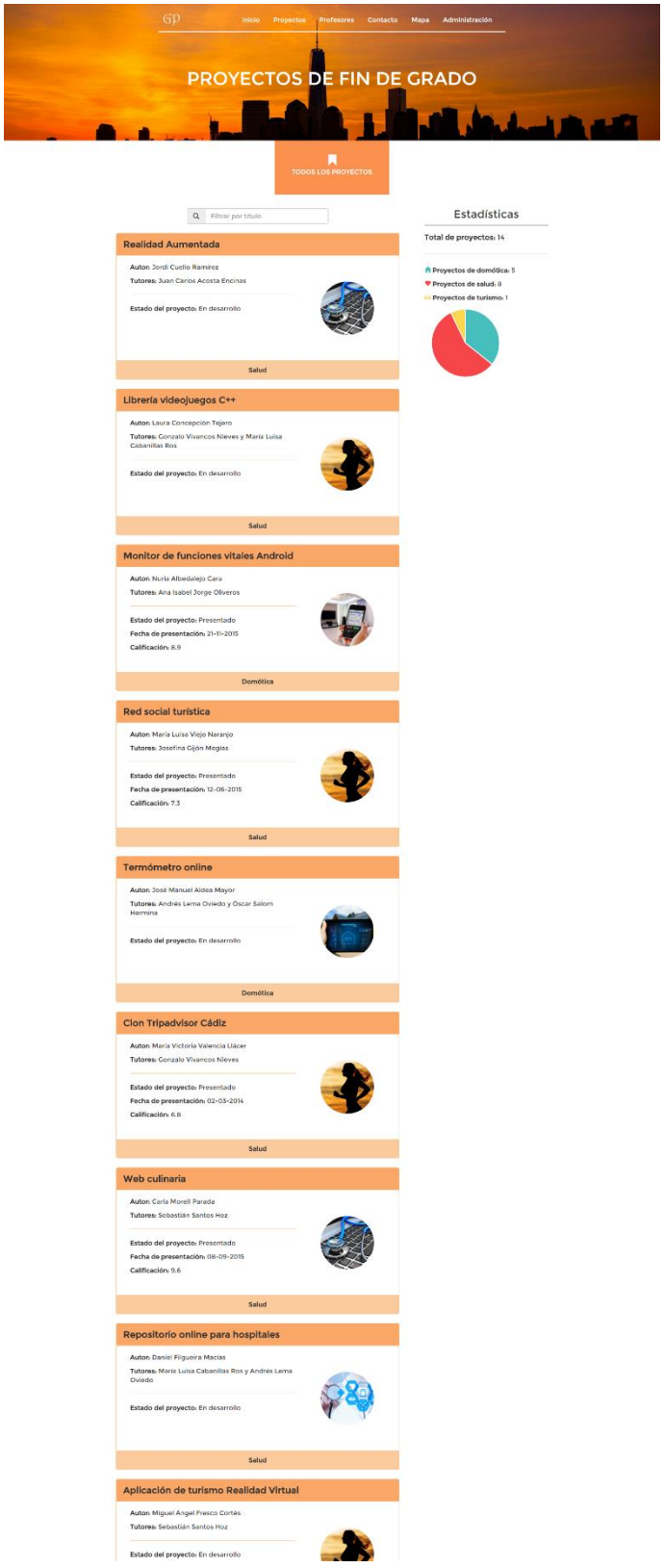

Figura 2. Proyectos

3.2.3. JavaScript

Importamos el fichero fuente de AngularJS, nuestro fichero principal *app.js* y un *plugin* para gráficas:

```
<script src="js/angular.js"></script>
<script src="js/app.js"></script>
<script src="js/Chart.js-master/Chart.js"></script>
```

Como hicimos en la página principal añadimos *ng-app* a la etiqueta *html* para inicializar la aplicación Angular y e indicamos el ámbito del controlador con *ng-controller:*

```
<html lang="es" ng-app="GestionProyectos">
<body ng-controller="ProjectsCtrl">
```

La primera funcionalidad de Angular que vemos en la estructura del HTML es la de filtrar una serie de resultados:

```
<input type="text" class="form-control" ng-model="title"
placeholder="Filtrar por título">

<div class="panel panel-default" ng-repeat="project in projects |
filter:{title: title}">
```

La directiva *ng-model* asocial el valor introducido en ese campo de entrada con el nombre indicado en el valor, en este caso “title”. Así, al hacer referencia a “title” desde algún otro lugar se obtendrá el valor introducido en el campo.

Existe la posibilidad de aplicar filtros sobre la fuente de datos utilizada en *ng-repeat.* Con el valor encadenado *filter:{title: title}* estamos indicando que sobre el conjunto de proyectos contenido en la variable *projects* se filtren aquellos cuyo atributo *title* (*project.title*) sean igual o contengan el valor contenido en la variable *title*, es decir, el valor introducido en el campo de texto.

Observamos a continuación de nuevo el HTML del panel de proyecto. Incluye la directiva *ng-repeat* así que habrá múltiples:

```
<div class="panel panel-default" ng-repeat="project in projects |
filter:{title: title}">
        <div class="panel-heading">{{ project.title }}</div>
        <div class="panel-body">
```

```
                <div class="col-md-8">
                        <p><strong>Autor: </strong>{{ project.studentName
}}</p>
                        <p><strong>Tutores: </strong>{{ project.tutor }}
<span ng-hide="!project.cotutor || project.cotutor.length === 0">y {{
project.cotutor }}</span></p>
                        <hr>
                        <p><strong>Estado del proyecto: </strong>{{
project.state }}</p>
                        <p ng-hide="!project.date || project.date.length ===
0"><strong>Fecha de presentación: </strong>{{ project.date }}</p>
                        <p ng-hide="!project.mark || project.mark.length ===
0"><strong>Calificación: </strong>{{ project.mark }}</p>
                </div>
                <div class="project-image-container col-md-4">
                        <img src="img/{{ project.img }}" alt="{{
project.imgDescription }}" class="img-circle">
                </div>
        </div>
        <div class="panel-footer">
                {{ project.topic }}
        </div>
</div>
```

Anteriormente ya explicamos el funcionamiento de *ng-repeat* y de las llaves *{{}}*. Pero se incluye una nueva directiva: *ng-hide*. Esta directiva hará que el elemento se oculte si se cumple la condición indicada en el valor con una expresión en JavaScript. Véase por ejemplo la primera de ellas:

```
<span ng-hide="!project.cotutor || project.cotutor.length === 0">y {{
project.cotutor }}</span>
```

El elemento <*span*> se ocultará si no está establecida la propiedad *cotutor* sobre la variable temporal *project* o si la longitud de *cotutor* es 0, es decir, si el proyecto no tiene cotutor, no mostrar esa información.

La variable *projects* se obtiene de manera similar a como se obtenía la lista de los seis últimos proyectos en la página de inicio. No obstante, aquí presentamos información adicional que no devuelve el servicio web, así que la generamos aleatoriamente.

Declaramos el controlador *ProjectsCtrl* e invocamos nuevamente al servicio de Angular *factory* para obtener el JSON de todos los proyectos y una vez se obtenga la

respuesta formamos el *array* con los proyectos, los cuales además de tener la información enviada por el servicio web tiene información autogenerada.

Por último, se crea un *array* con los datos necesarios por el *plugin* del gráfico circular para generarlo y se dibuja tal y como indica la documentación de *Chart.js*:

```
app.controller('ProjectsCtrl', function($scope, ws) {

        ws.getAllStudents().then(function (data) {
                var projects = [];

                var topics = {
                        "Domótica": {
                                "count": 0,
                                "path": "domotica_?.png",
                                "alt": "Imagen icónica representativa de la
domótica"
                        },
                        "Salud": {
                                "count": 0,
                                "path": "salud_?.png",
                                "alt": "Imagen icónica representativa de la
sulud"
                        },
                        "Turismo": {
                                "count": 0,
                                "path": "turismo_?.png",
                                "alt": "Imagen icónica representativa del
turismo"
                        }
                }
                var keys = Object.keys(topics);

                $.each(data, function(key, student) {
                        var topic = keys[randomInt(0, keys.length-1)];

                        topics[topic]["count"]++;

                        projects.push({
                                "studentName": (student.nombre + " " +
student.primerApellido + " " + student.segundoApellido).trim(),
                                "title": student.tituloProyecto,
                                "tutor": student.tutor1,
                                "cotutor": student.tutor2,
```

```
                                    "state":
student.estadoProyecto.charAt(0).toUpperCase() +
student.estadoProyecto.slice(1),
                                    "date": student.fechaPresentacionProyecto,
                                    "mark": student.calificacionProyecto,
                                    "topic": topic,
                                    "img": topics[topic]["path"].replace("?",
randomInt(1, 3)),
                                    "imgDescription": topics[topic]["alt"]
                            });
                });

                $scope.homeAutomation = topics["Domótica"]["count"];
                $scope.health = topics["Salud"]["count"];
                $scope.tourism = topics["Turismo"]["count"];

                $scope.projects = projects;

                context = $("#chart").get(0).getContext("2d");
                var data = [
                    {
                        value: topics["Domótica"]["count"],
                        color: "#46BFBD",
                        highlight: "#5AD3D1",
                        label: "Domótica"
                    },
                        {
                        value: topics["Salud"]["count"],
                        color:"#F7464A",
                        highlight: "#FF5A5E",
                        label: "Salud"
                    },
                    {
                        value: topics["Turismo"]["count"],
                        color: "#FED750",
                        highlight: "#FEE36B",
                        label: "Turismo"
                    }
                ];

                var chart = new Chart(context).Pie(data);
        });

});
```

3.3. Profesores

3.3.1. Estructura

La sección superior tiene una estructura idéntica a la página de los proyectos, así como el comienzo de la sección principal, donde también añadimos una pestaña informativa.

El segundo hijo de la sección principal es la sección de profesores. En esta sección primero tenemos un <*article*> con la descripción de esta sección, los profesores del departamento. A continuación, una sección lateral <*aside*> con el logo de la UCA, enlazando con la web del departamento. Por último, otro <*article*> que contiene una tabla con los profesores. Para ello se utiliza la etiqueta <*table*>¸ para indicar el cuerpo de las cabeceras <*thead*> y el cuerpo principal de la tabla <*tbody*>. Cada fila de la tabla se representa con <*tr*> y cada celda, si es de la cabecera <*th*> y si es una celda corriente <*td*>.

```
<section id="professors-section" class="row">
        <article id="professors-description" class="col-md-8">
                <h2>Nuestros profesores</h2>
                <p>Los profesores de nuestro departamento son profesionales
de confianza de la Universidad de Cádiz. Imparten docencia en la especialidad
de Sistemas de la Información y Tecnologías de la Información.</p>
                <p>Su campo de conocimiento domina diversas áreas, entre
otras, tecnologías web <em>back-end</em> y <em>front-end</em>, servicios web,
sistemas hipermedia, administración de servidores y de redes, seguridad y
calidad de sistermas, tratamiento de datos, organización...</p>
                <p>Sus trabajos de investigación se encuentran entre los más
destacados de la Escuela Superior de Ingeniería y destacan por el uso de las
tecnologías más innovaduras.</p>
        </article>
        <aside id="aside" class="col-md-4">
                <a href="http://departamentos.uca.es/C137/" target="_blank">
                        <img src="img/logo_uca.png" alt="Logo de la UCA">
                </a>
        </aside>
        <article id="professors" class="col-md-12">
                <h2>Tabla de profesores</h2>
                <table id="professors-table" class="table table-hover">
                        <thead>
                                <tr>
                                        <th>Nombre</th>
                                        <th>Puesto</th>
```

```
                                        <th>Teléfono</th>
                                        <th>Correo electrónico</th>
                                </tr>
                        </thead>
                        <tbody>
                                <tr ng-repeat="professor in professors">
                                        <td>{{ professor.name }}</td>
                                        <td>{{ professor.position }}</td>
                                        <td><a href="tel:+34{{
professor.phone }}">{{ professor.phoneFormat }}</a></td>
                                        <td><a href="mailto:{{
professor.email }}">{{ professor.email }}</a></td>
                                </tr>
                        </tbody>
                </table>
        </article>
</section>
```

La siguiente sección es la sección de los perfiles, que contiene un <*div*> con un panel y en éste una imagen <*img*> con el avatar del profesor y otro <*div*> a modo de pie con su nombre:

```
<section id="profiles" class="col-md-8">
        <div class="row">
                <h2>Perfiles</h2>
                <div class="col-md-4" ng-repeat="profile in profiles">
                        <div class="panel panel-default">
                                <div class="panel-body">
                                        <img class="img-circle" src="img/{{
profile.img }}" alt="{{ profile.description }}">
                                </div>
                                <div class="panel-footer">
                                        <span>{{ profile.name }}</span>
                                </div>
                        </div>
                </div>
        </div>
</section>
```

La última parte de la sección principal es una sección lateral con las estadísticas de la participación de los profesores en distintos proyectos. Es idéntica a la sección lateral de la página de los proyectos:

```
<aside id="ranking" class="col-md-4">
        <h3 class="aside-header">Profesores más activos</h3>
        <ul class="list-unstyled">
                <li ng-repeat="professor in ranking">
                        <span><strong>{{ professor.name }}, </strong>{{
professor.projectCount }} proyectos</span>
                </li>
        </ul>
        <div class="col-md-12">
                <canvas id="chart" width="300px" height="150px"></canvas>
        </div>
</aside>
```

Lo último de la página es el pie <*footer*> anteriormente explicado y usado en todas las páginas.

3.3.2. Estilo

Igual que en las demás páginas, usamos Bootstrap, *app.css* y un fichero de estilos únicamente para esta página: *profesores.css.*

Siguiendo el estilo de sección superior con una imagen de fondo, lo estilamos de ese modo:

```
#upper-section {
        height: 300px;
        background-image: url(../img/background-profesores.jpg);
        background-size: cover;
}
```

La gama de colores de esta página es morado, así pues le damos a la pestaña informativa un fondo morado con *background-color*:

```
.iconic-information-block {
        margin: 0 auto;
        float: none;
        background-color: #563dae;
        color: #fff;
        text-transform: uppercase;
}
```

Al logo de la UCA en el lateral le damos un ancho fijo y lo centramos:

```
#aside {
        text-align: center;
        margin-top: 30px;
}

#aside img {
        width: 150px;
}
```

Para la tabla hemos usado la clase *table* de Bootstrap. Sobreescribimos algunos estilos propios de Bootstrap con estilos propios para la tabla que encajen con el color de la página:

```
#professors-table a {
        color: #8470BD;
}

#professors-table thead tr th {
        border-bottom: 2px solid #C5BAF5;
}

#professors-table tbody tr td {
        border-top: 1px solid #C5BAF5;
}

#professors-table tbody tr:hover td {
        background-color: #dfd9f5;
}
```

Para los perfiles, que usa Bootstrap panel, añadimos una separación a la cabecera con *margin-bottom,* fijamos un tamaño particular a la imagen del avatar, y modificamos el pie del panel añadiendo un borde redondeado y usamos las propiedades CSS3 *display: flex; justify-content: center; flex-direction: column;* para centrar el texto del pie verticalmente.

```
#profiles {
        text-align: center;
}

#profiles h2 {
```

```
        margin-bottom: 30px;
}

#profiles .panel-default {
        border: 1px solid transparent;
}

#profiles .panel-footer {
        border: 1px solid #8f73cc;
        background-color: #8f73cc;
        color: #fff;
        border-radius: 3px;
        height: 62px;
        padding-top: 10px;
        display: flex;
        justify-content: center;
        flex-direction: column;
}

#profiles img {
        width: 100px;
        height: 100px;
}
```

Finalmente, ajustamos los márgenes y *padding* de la sección lateral con la estadística de participación:

```
#ranking {
        margin: 30px auto;
        padding-left: 40px;
}

#ranking hr {
        border-color: #ddd;
}

#ranking li {
        padding: 5px 0px;
}

#chart {
        padding-left: 0px;
}
```

La Figura 3 muestra cómo luce la página de profesores tras aplicar los estilos.

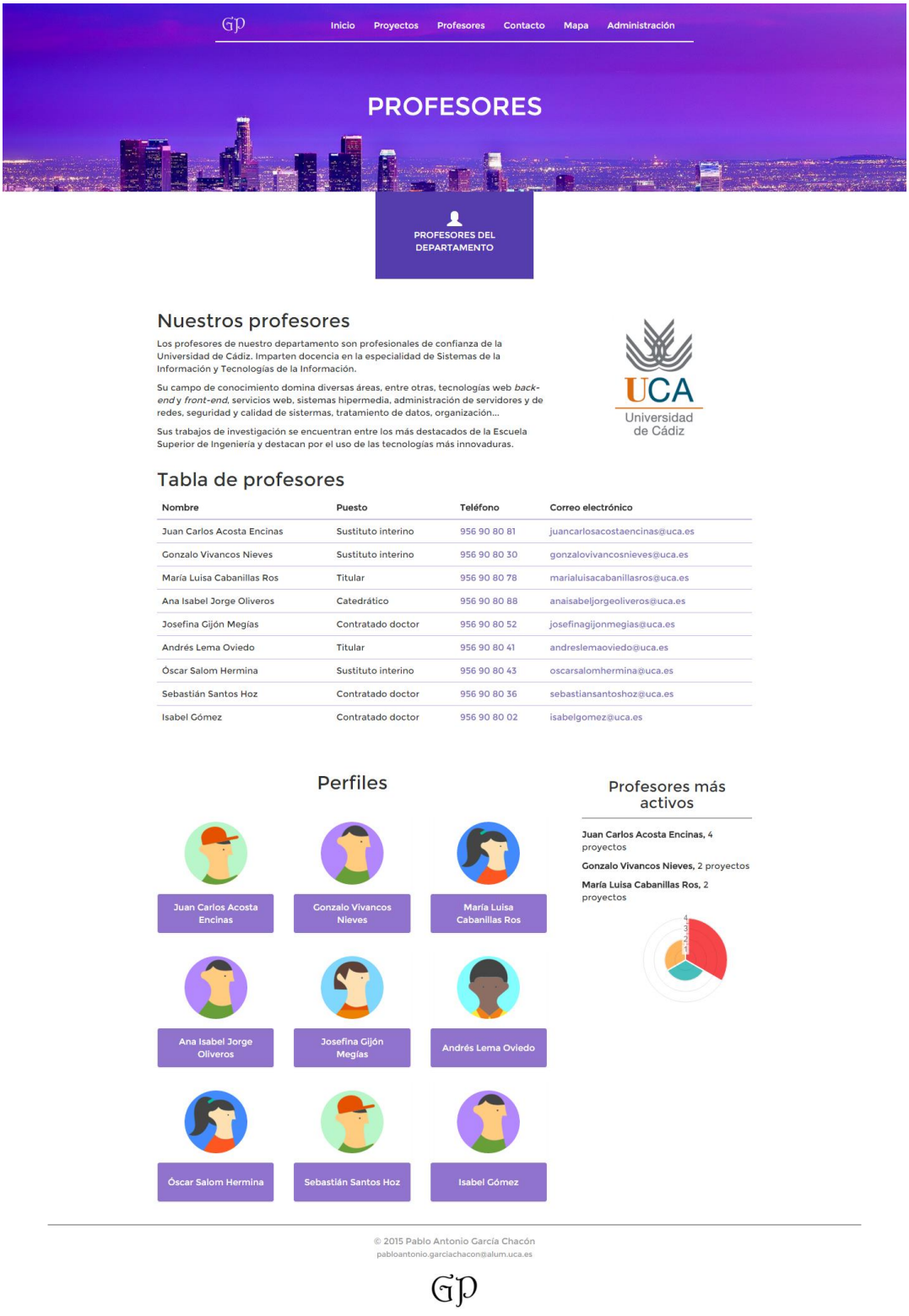

Figura 3. Profesores

3.3.3. JavaScript

Importamos los ficheros fuente de Angular y Chart.js, así como nuestro *app.js.* Al igual que hicimos anteriormente, ligamos el documento a Angular con *ng-app* y a la etiqueta *body* le añadimos *ng-controller* con el valor de "ProfessorsCtrl".

Usamos de nuevo *ng-repeat* para iterar sobre una lista de profesores y mostrar una fila en la tabla con su información:

```
<tr ng-repeat="professor in professors">
        <td>{{ professor.name }}</td>
        <td>{{ professor.position }}</td>
        <td><a href="tel:+34{{ professor.phone }}">{{ professor.phoneFormat
}}</a></td>
        <td><a href="mailto:{{ professor.email }}">{{ professor.email
}}</a></td>
</tr>
```

En *app.js* añadimos el controlador *ProfessorsCtrl* y en su cuerpo hacemos la llamada a *ws.getAllEstudiantes()* para obtener el JSON con los estudiantes del servicio web, y encadenamos *then* para definir la lógica que siga a la respuesta del servicio:

```
app.controller('ProfessorsCtrl', function($scope, ws) {

        ws.getAllStudents().then(function (data) {
        }
});
```

La única información obtenida del servicio acerca de los profesores es su nombre. Por tanto la información de su puesto, teléfono y correo electrónico debe ser autogenerada de forma aleatoria:

```
var professorsTmp = [];

$.each(data, function(key, student) {
        professorsTmp.push(student.tutor1);
        if (student.tutor2 && student.tutor2.len !== 0)
                professorsTmp.push(student.tutor2);
});

var professors = {};
var profiles = [];
```

```
var positions = [
        "Catedrático",
        "Titular",
        "Contratado doctor",
        "Ayudante doctor",
        "Sustituto interino"
];

$.each(professorsTmp, function(key, professor) {
        if (professor in professors)
                professors[professor]["count"] =
professors[professor]["count"] + 1;
        else {
                var d1 = randomInt(0, 9).toString();
                var d2 = randomInt(0, 9).toString();

                professors[professor] = {};

                professors[professor] = {
                        "name": professor,
                        "position": positions[randomInt(0, positions.length-
1)],
                        "phoneFormat": "956 90 80 " + d1.toString() +
d2.toString(),
                        "phone": "9569080" + d1 + d2,
                        "email": buildEmail(professor),
                        "count": 1
                };

                var avatar = "avatar-?.png";
                var rnd = randomInt(1, 14);
                if (rnd < 10) avatar = avatar.replace("?", "0" + rnd);
                else avatar = avatar.replace("?", rnd);

                profiles.push({
                        "name": professor,
                        "img": avatar,
                        "description": "Foto de " + professor
                });
        }
});
```

A continuación definimos la lógica que forma el *array* de los tres profesores con mayor participación en proyectos de fin de grado y añadimos toda la información al *$scope* para que sea accesible desde el HTML:

```
var projectParticipation = new Array();

$.each(professors, function(key, professor) {
        projectParticipation.push({
                name: professor["name"],
                projectCount: professor["count"]
        });
});

projectParticipation = projectParticipation.sort(function (a, b) {
        a = a.projectCount;
        b = b.projectCount;
        return a > b ? -1 : (a < b ? 1 : 0);
});

var ranking = [];
for (i = 0; i < 3; i++) {
        ranking.push({
                "name": projectParticipation[i].name,
                "projectCount": projectParticipation[i].projectCount
        });
}

$scope.professors = professors;
$scope.profiles = profiles;
$scope.ranking = ranking;
```

Por último, generamos el gráfico con la información obtenida:

```
context = $("#chart").get(0).getContext("2d");
var data = [
    {
        value: ranking[0].projectCount,
        color: "#F7464A",
        highlight: "#FF5A5E",
        label: ranking[0].name
    },
        {
        value: ranking[1].projectCount,
```

```
        color:"#46BFBD",
        highlight: "#5AD3D1",
        label: ranking[1].name
    },
    {
        value: ranking[2].projectCount,
        color: "#FDB45C",
        highlight: "#FFC870",
        label: ranking[2].name
    }
];

var chart = new Chart(context).PolarArea(data);
```

3.4. Contacto

3.4.1. Estructura

La sección superior de la página de contacto es igual que en las anteriores: una altura fija, contiene el menú superior y el título de la página.

La sección principal contiene otra sección con un formulario para enviar un mensaje. En HTML la etiqueta utilizada para los formularios se trata de *<form>*. Cabe destacar el atributo *action* que indica el recurso adonde se enviará la información contenida en el formulario. Otro atributo importante es *method,* cuyo valor puede ser *get* (por defecto) o *post*, correspondientes a los métodos HTTP.

La información del formulario está organizada usando la etiqueta *<fieldset>* y para el título del formulario se usa *<legend>*.

El formulario contiene dos tipos de campos de entrada distintos: una entrada de texto representada por *<input type="text">*, y una caja de texto (pensada para textos de mayor longitud) con un número de dimensión de partida usando la etiqueta *<textarea col="30" rows="10">*. Cabe mencionar el atributo *name* en los campos de entrada: al enviar un formulario se envía la información introducida en los campos en forma de conjunto de claves-valores, el atributo *name* corresponde a la clave, y el valor del campo al valor.

```
<section id="contact">
        <h2>Cómo contactar</h2>
        <p>Puedes enviar un correo a la dirección <a
href="mailto:pabloantonio.garciachacon@alum.uca.es">pabloantonio.garciachacon
@alum.uca.es</a> o rellenar y enviar el siguiente formulario.</p>
        <form action="contactar.html" method="post">
                <fieldset>
                        <legend>Envía un mensaje</legend>
                        <div class="form-group">
                                <label for="subject">Asunto</label>
                                <input type="text" id="subject"
name="subject" class="form-control"><br>
                        </div>
                        <div class="form-group">
                                <label for="body">Cuerpo</label>
                                <textarea id="body" name="body" class="form-
control" cols="30" rows="10"></textarea>
                        </div>
```

```
                        <button type="submit" class="btn btn-primary btn-
lg">
                                <span class="glyphicon glyphicon-
envelope"></span>
                        </button>
                </fieldset>
        </form>
</section>
```

Por último, en esta sección se incluye una sección lateral con el mapa indicando la ubicación de la Escuela Superior de Ingeniería:

```
<aside class="col-md-4">
        <div class="row">
                <h2>Ubicación</h2>
                <address>Nos puedes encontrar en la Escuela Superior de
Ingeniería de la Universidad de Cádiz.<br/>Avenida Universidad de Cadiz, 10,
11519 Puerto Real, Cádiz.</address>
                <div id="map"></div>
        </div>
</aside>
```

La última parte de la estructura es el pie de página común a todas ellas.

3.4.2. Estilo

El primer estilo que definimos en *contactar.css* es el de la sección superior, dándole una altura fija y una imagen de fondo:

```
#upper-section {
        height: 300px;
        background-image: url(../img/background-contactar.jpg);
        background-size: cover;
}
```

Damos algo de espacio a la sección principal por abajo y por arriba:

```
#main-section {
        margin-top: 50px;
        margin-bottom: 50px;
```

```
}
```

A la sección de contacto le damos un *padding* inferior y superior y resaltamos la cabecera y le añadimos un borde inferior:

```
#contact {
        padding: 0px 40px;
}

#contact h2 {
        font-size: 36px;
        border-bottom: 2px solid #aaa;
        padding-bottom: 10px;
}
```

Se centra el formulario con *margin: 0 auto*, y el *fieldset* y *legend* se modifican bastante con respecto a su aspecto por defecto dado por Bootstrap: se le añade un borde redondeado, *padding* y un color de fondo gris al *fieldset*, y a *legend* se le quita el borde inferior. A los campos se les da un color de fondo gris más oscuro para contrastar con el fondo del *fieldset*. El botón de enviar también se modifica con respecto a su aspecto en Bootstrap, se aumenta el tamaño de icono y se centra horizontalmente:

```
#contact form {
        margin: 0 auto;
}

#contact form fieldset {
        border: 1px solid #aaa;
        padding: 15px 30px;
        border-radius: 5px;
        background-color: #eee;
}

#contact form fieldset legend {
        width: inherit;
        padding: 0px 10px;
        border-bottom: none;
}

#contact form input, #contact form textarea {
        background-color: #e5e5e5;
}
```

```
#contact form textarea {
        resize: none;
}

#contact form button {
        display: block;
        width: 30%;
        margin: 0 auto;
        font-size: 24px;
        outline: none;
}
```

El formulario usa muchas clases de Bootstrap, como *form-control* para agrupar un *label* y su campo correspondiente, o *btn* y sus derivados para el botón de enviar.

En la Figura 4 se puede observar el aspecto final de la página de contacto.

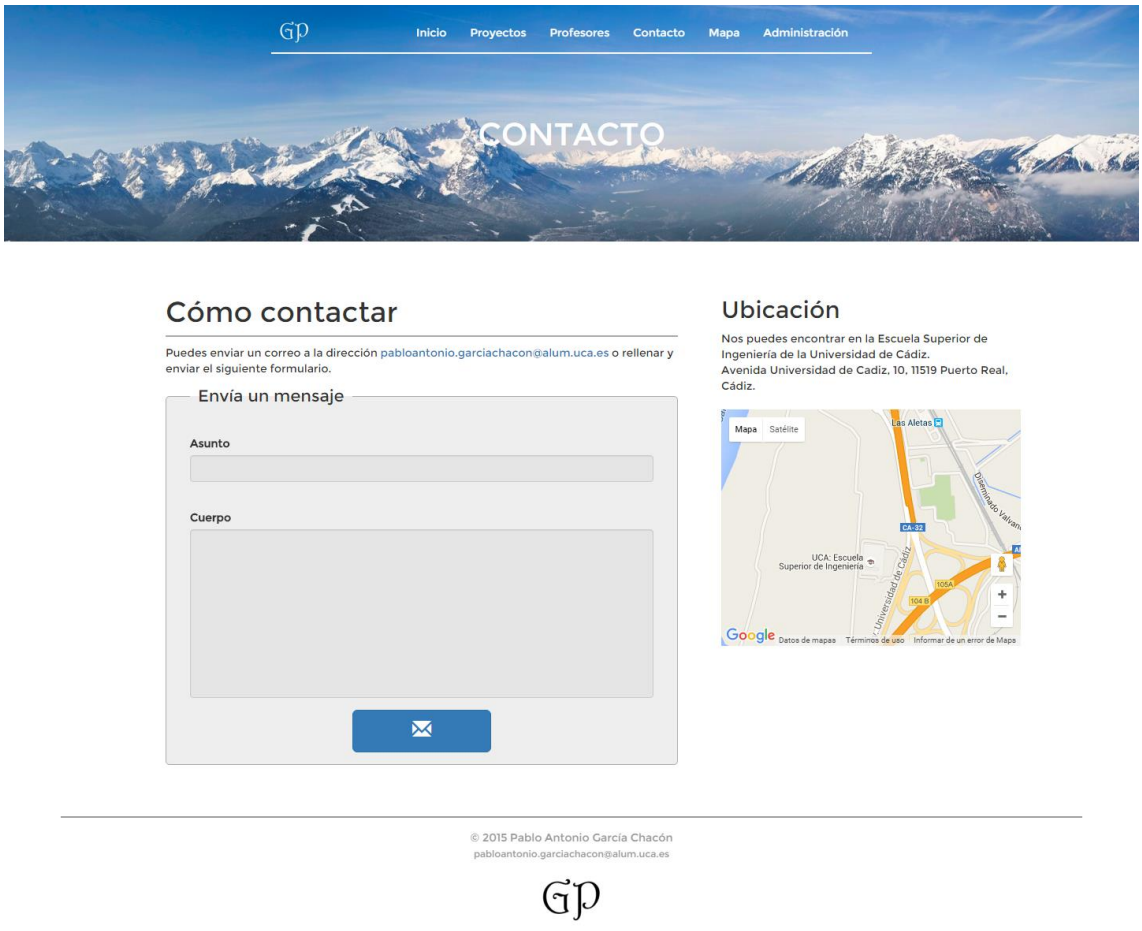

Figura 4. Página de contacto

3.4.3. JavaScript

El único JavaScript utilizado en esta página es el mapa de Google, y su inclusión es exactamente igual que lo explicado en la página principal.

3.5. Administración

La página de administración es la última página del sitio y a través de ella podemos hacer interacciones *CRUD* los proyectos invocando las operaciones del servicio web a través de Angular.

3.5.1. Estructura

Abrimos el fichero *administración.html* y escribimos un esquema HTML5 como en todos los documentos HTML anteriores, y tiene también una sección superior idéntica a las demás páginas.

También hay una sección principal, donde primero añadimos una pestaña informativa y, a continuación, una sección que contiene las acciones que podemos hacer usando el servicio, usando la etiqueta <*section*>.

El primer hijo de la sección de acciones tiene un bloque <*div*> descriptivo del propósito de la página, y un título usando la cabecera <*h2*>.:

```
<section id="actions-container">
        <div id="actions-header-container">
                <h2>Elige qué acción quieres realizar</h2>
                <p>Con el panel de administración puedes crear un nuevo
proyecto, modificar uno existente, o eliminarlo.</p>
        </div>
</section>
```

El siguiente y último hijo de la sección de acciones es un contenedor de las propias acciones. Hay cuatro acciones en total: crear, actualizar, eliminar y buscar. Para cada una de las cuales se usa un bloque <*div*> que contiene un icono <*span*> y un texto descriptivo de la acción usando un párrafo con la etiqueta <*p*>. En la siguiente sección se explicará cómo se consigue un bloque <*div*> con aspecto de bloque circular:

```
<div id="actions" class="container small-container">
        <div class="action-container col-md-3 col-sm-6">
                <div class="iconic-information-block action create" data-
toggle="modal" data-target="#modal-create" ng-click="resetFormState()">
                        <div class="action-inner-container">
                                <span class="glyphicon glyphicon-ok"></span>
                                <p>Crear</p>
                        </div>
                </div>
        </div>
```

```
        <div <cl></cl>ass="action-container col-md-3 col-sm-6">
                <div class="iconic-information-block action update" data-
toggle="modal" data-target="#modal-update" ng-click="resetFormState()">
                        <div class="action-inner-container">
                                <span class="glyphicon glyphicon-
refresh"></span>
                                <p>Actualizar</p>
                        </div>
                </div>
        </div>
        <div class="action-container col-md-3 col-sm-6">
                <div class="iconic-information-block action delete" data-
toggle="modal" data-target="#modal-delete" ng-click="resetFormState()">
                        <div class="action-inner-container">
                                <span class="glyphicon glyphicon-
remove"></span>
                                <p>Eliminar</p>
                        </div>
                </div>
        </div>
        <div class="action-container col-md-3 col-sm-6">
                <div class="iconic-information-block action search" data-
toggle="modal" data-target="#modal-search" ng-click="resetFormState()">
                        <div class="action-inner-container">
                                <span class="glyphicon glyphicon-
remove"></span>
                                <p>Buscar</p>
                        </div>
                </div>
        </div>
</div>
```

Al final de la página tenemos el <*footer*> que hemos usado en todas las páginas.

Además de los elementos mencionados, está página incluye cuatro distintos *modals*. Estos son elementos de Bootstrap ocultos en un principio pero que se muestran por medio de JavaScript al pulsar un botón o al hacer una acción determinada que llame a la función que haga mostrar al *modal*.

Para cada acción usamos una etiqueta <*section*> para añadir una sección y el hijo inmediato es el *modal*. Cada *modal* tiene un contenido que se divide en la cabecera, con el título de la acción, y el cuerpo, que contendrá un formulario o bien los resultados

de enviar el mismo. Por ejemplo, la sección y el *modal* utilizados para la acción de **crear**, que es igual que en el resto de los casos, se estructura como sigue:

```
<section id="modal-create-container">
	<div class="modal fade" id="modal-create" role="dialog" aria-
labelledby="modal-create-label">
		<div class="modal-dialog" role="dialog">
			<div class="modal-content">
				<div class="modal-header">
					<button type="button" class="close" data-
dismiss="modal" aria-label="Cerrar">
						<span aria-
hidden="true">&times;</span>
					</button>
					<h4 class="modal-title" id="modal-create-
label">Crear proyecto</h4>
				</div>
				<div ng-hide="!error || error.length === 0"
class="modal-error">
					<h4 class="modal-title">{{ error }}</h4>
				</div>
				<div class="modal-body">
				</div>
			</div>
		</div>
	</div>
</section>
```

El <*div*> con la clase *modal-error* aparecerá sobre la cabecera del *modal* si se da un error en el servidor indicando un mensaje de error descriptivo.

En el cuerpo del *modal* (*div* con la clase *modal-body*) hay dos hijos: el formulario y un contenedor <*div*> con los resultados. Solamente se mostrará uno de ellos a la vez gracias a AngularJS, el funcionamiento se explicará en el apartado de JavaScript.

Para el formulario usamos la etiqueta <*form*> y no establecemos ningún *action* o *method* ya que Angular se encargará de esto. Cada campo del formulario está envuelto en un <*div*> padre con la clase de Bootstrap *input-group* para aplicar los estilos de Bootstrap. Para cada campo se usa un *label* que indica el nombre del campo.

Veamos lo explicado en el primer campo, el nombre del estudiante, para el que se usa un <*input type=”text”*> porque un nombre es texto. El atributo *required* indica que el campo es obligatorio y se impedirá el envío del formulario si no se rellena:

```
<form ng-hide="resultProject" id="create" name="create" ng-
submit="submitCreateForm(create)" class="form-horizontal" novalidate>
		<div class="form-group">
```

```
                <label for="name" class="col-md-3 control-
label">Nombre</label>
                <div class="col-md-9">
                        <input required type="text" ng-model="project.name"
ng-pattern="/^[a-z\u00E0-\u00FC ]+$/i" name="name" class="form-control">
                </div>
        </div>
</form>
```

Los siguientes campos siguen la misma estructura HTML y también usan *inputs* de *type=”text”* porque se tratan del primer apellido y segundo apellido del estudiante:

```
<div class="form-group">
       <label for="firstLastName" class="col-md-3 control-label">Primer
apellido</label>
       <div class="col-md-9">
              <input required type="text" ng-model="project.firstLastName" ng-
pattern="/^[a-z\u00E0-\u00FC ]+$/i" name="firstLastName" class="form-
control">
       </div>
</div>
<div class="form-group">
       <label for="secondLastName" class="col-md-3 control-label">Segundo
apellido</label>
       <div class="col-md-9">
              <input type="text" ng-model="project.secondLastName" ng-
pattern="/^[a-z\u00E0-\u00FC ]+$/i" name="secondLastName" class="form-
control">
       </div>
</div>
```

El siguiente campo, e-mail, utiliza un *input* de *type=”email”*, es un nuevo tipo introducido por HTML5 y, en algunos navegadores a partir de determinadas versiones, será validada la corrección del patrón del e-mail introducido:

```
<div class="form-group">
        <label for="email" class="col-md-3 control-label">Correo
electrónico</label>
        <div class="col-md-9">
                <input required type="email" ng-model="project.email"
name="email" class="form-control" email-uca>
        </div>
</div>
```

El campo título, donde introducir el título del proyecto, también utiliza tipo texto:

```
<div class="form-group">
        <label for="title" class="col-md-3 control-label">Título del
proyecto</label>
        <div class="col-md-9">
                <input required type="text" ng-model="project.title"
name="title" class="form-control">
        </div>
</div>
```

El campo a continuación de título se trata un menú desplegable donde elegir una opción de entre las distintas categorías disponibles para asignar al proyecto (aunque no se guardará en la base de datos ya que el servicio no cubre este atributo). Para mostrar un campo de entrada en forma de menú desplegable se utiliza la etiqueta *<select>* y cada una de las opciones son elementos hijos con la etiqueta *<option>* donde su atributo *value* indica el valor que tiene la opción y es, por tanto, el valor que tome el campo de entrada cuando se seleccione dicha opción:

```
<div class="form-group">
        <label for="topic" class="col-md-3 control-label">Temática</label>
        <div class="col-md-9">
                <select ng-model="project.topic" name="topic" class="form-
control">
                        <option value="Domótica">Domótica</option>
                        <option value="Salud">Salud</option>
                        <option value="Turismo">Turismo</option>
                </select>
        </div>
</div>
```

Los siguientes dos campos, tutor y cotutor, donde se introducen los nombres completos del tutor y del tutor secundario, si lo hubiese, son campos de texto:

```
<div class="form-group">
        <label for="tutor" class="col-md-3 control-label">Tutor</label>
        <div class="col-md-9">
                <input required type="text" ng-model="project.tutor" ng-
pattern="/^[a-z\u00E0-\u00FC ]+$/i" name="tutor" class="form-control">
        </div>
</div>
```

```
<div class="form-group">
        <label for="cotutor" class="col-md-3 control-label">Cotutor</label>
        <div class="col-md-9">
                <input type="text" ng-model="project.cotutor" ng-
pattern="/^[a-z\u00E0-\u00FC ]+$/i" name="cotutor" class="form-control">
        </div>
</div>
```

El siguiente campo es el de estado de proyecto. Un proyecto puede tener dos estados: presentado o en desarrollo. Este tipo de campo utiliza botones excluyentes, lo que significa que al seleccionar uno de ellos se deseleccionará el otro. Para una entrada de tipo botón excluyente o *radio* se utiliza la etiqueta *<input type="radio">*, tantos como alternativas haya, y para que identifiquen el mismo dato de entrada y sean mutuamente excluyentes han de tener el mismo nombre (atributo *name*), de lo contrario se tratarán como campos distintos:

```
<div class="form-group">
        <label for="state" class="col-md-3 control-label">Estado del
proyecto</label>
        <div class="col-md-9">
                <label class="radio-inline">
                        <input type="radio" name="state" value="presentado"
ng-model="project.state" ng-change="changeState(project.state)"> Presentado
                </label>
                <label class="radio-inline">
                        <input type="radio" name="state" value="en
desarrollo" ng-model="project.state" ng-change="changeState(project.state)"
ng-click="clearPresentedAttributes()"> En desarrollo
                </label>
        </div>
</div>
```

El campo siguiente es el campo fecha de presentación, y es de tipo texto (usamos *<input type="text">*. El valor de la fecha no se introduce desde el teclado sino que al pulsar sobre el campo de entrada se muestra un calendario mediante un *plugin* JavaScript, que se explicará en la sección correspondiente. Existe un tipo de *input* para fechas usando *type="date"*, sin embargo, algunos navegadores de los más usados no soportan este tipo así que se produce incompatibilidad entre navegadores al intentar usar este tipo de campo, por eso finalmente se ha de usar tipo texto:

```
<div class="form-group" ng-hide="selectedState === 'en desarrollo'">
```

```
        <label for="date" class="col-md-3 control-label">Fecha de
presentación</label>
        <div class="col-md-9">
                <input type="text" ng-model="project.date" name="date"
class="form-control" date-field>
        </div>
</div>
```

El último campo de entrada es el campo de calificación. Es de tipo numérico y usamos el nuevo tipo de dato HTML5 *type="number"* lo que impide al usuario introducir por teclado información distinta a números. Además, para la mayoría de los navegadores se muestra un elemento de interacción con el usuario para aumentar o reducir en una unidad el valor introducido (o si no existe ninguno empezando por 0), mediante flechas hacia arriba y hacia abajo o similar. Usamos los atributos *min* y *max* para indicar el rango permitido para el valor introducido:

```
<div class="form-group" ng-hide="selectedState === 'en desarrollo'">
        <label for="mark" class="col-md-3 control-
label">Calificación</label>
        <div class="col-md-9">
                <input type="number" ng-model="project.mark" name="mark"
min="0" max="10" class="form-control">
        </div>
</div>
```

Los últimos elementos del formulario son los botones de restablecer (vuelve al estado inicial de formulario vacío) y de enviar el formulario. Para el primero se utiliza la etiqueta *<button type="button">* porque es un botón genérico y para el botón de envío se utiliza *<button type="submit">* porque queremos que el navegador envíe el `formulario.`

```
<div class="button-container">
        <button type="button" class="btn btn-default" ng-
click="reset(create)">Restablecer</button>
        <button type="submit" class="btn btn-info">Crear</button>
</div>
```

Habiendo explicado el primer hijo que contiene el cuerpo del *modal* pasamos a ver el bloque * * segundo hijo del *modal* que aparece una vez se envíe el formulario y se obtenga una respuesta satisfactoria del servidor.

Este bloque simplemente contiene otros bloques * * a su vez, mostrando un mensaje de éxito y, a continuación, separado por un *<hr>*, información relativa a la ejecución con éxito de la acción realizada. En este caso, la información del estudiante creado:

```
<div ng-show="resultProject" class="small-container result">
        <div ng-show="success" class="info">
                <div class="alert alert-success" role="alert">{{ success
}}</div>
        </div>
        <div class="row">
                <hr>
        </div>
        <div class="row">
                <p><strong>Título: </strong>{{ resultProject.title }}</p>
                <p><strong>Autor: </strong>{{ resultProject.studentName
}}</p>
                <p><strong>Tutores: </strong>{{ resultProject.tutor }} <span
ng-hide="!resultProject.cotutor || resultProject.cotutor.length === 0">y {{
resultProject.cotutor }}</span></p>
                <hr>
                <p><strong>Estado del proyecto: </strong>{{
resultProject.state }}</p>
                <p ng-hide="!resultProject.date || resultProject.date.length
=== 0"><strong>Fecha de presentación: </strong>{{ resultProject.date }}</p>
                <p ng-hide="!resultProject.mark || resultProject.mark.length
=== 0"><strong>Calificación: </strong>{{ resultProject.mark }}</p>
        </div>
</div>
```

La siguiente acción, **actualizar**, recordamos que es también uno de los cuatro *modals* utilizados para cada acción y su estructura ya la hemos explicado. El contenido del modal es muy parecido al de la acción crear: tenemos un formulario con menos campos que el formulario de crear pues solamente se pueden actualizar algunos valores, y un *div* con el resultado de la actualización, los datos del estudiante ya actualizado:

```
<section id="modal-update-container">
        <div class="modal fade" id="modal-update" role="dialog" aria-
labelledby="modal-update-label">
                <div class="modal-dialog" role="dialog">
                        <div class="modal-content">
                                <div class="modal-header">
```

```
                                        <button type="button" class="close"
data-dismiss="modal" aria-label="Cerrar">
                                                <span aria-
hidden="true">&times;</span>
                                        </button>
                                        <h4 class="modal-title" id="modal-
update-label">Actualizar proyecto</h4>
                                </div>
                                <div ng-hide="!error || error.length === 0"
class="modal-error">
                                        <h4 class="modal-title">{{ error
}}</h4>
                                </div>
                                <div class="modal-body">
                                        <form ng-hide="resultProject"
id="update" name="update" ng-submit="submitUpdateForm(update)" class="form-
horizontal" novalidate>
                                                <div class="form-group">
                                                        <label for="key"
class="col-md-2 control-label">Clave</label>
                                                        <div class="col-md-
10">
                                                                <input
required type="text" ng-model="project.key" name="key" class="form-control"
placeholder="Clave del estudiante">
                                                                <div ng-
show="update.$submitted || update.key.$touched">

        <span class="help-block" ng-
show="update.key.$error.required">Introduce la clave de estudiante</span>
                                                                </div>
                                                        </div>
                                                </div>
                                                <div class="row">
                                                        <hr>
                                                </div>
                                                <div class="form-group">
                                                        <label for="title"
class="col-md-3 control-label">Título del proyecto</label>
                                                        <div class="col-md-
9">
                                                                <input
type="text" ng-model="project.title" name="title" class="form-control">
                                                        </div>
```

```
                                                    </div>
                                                    <div class="form-group">
                                                            <label for="state"
class="col-md-3 control-label">Estado del proyecto</label>
                                                            <div class="col-md-
9">
                                                                    <label
class="radio-inline">

        <input type="radio" name="state" value="presentado" ng-
model="project.state" ng-change="changeState(project.state)"> Presentado
                                                                    </label>
                                                                    <label
class="radio-inline">

        <input type="radio" name="state" value="en desarrollo" ng-
model="project.state" ng-change="changeState(project.state)" ng-
click="clearPresentedAttributes()"> En desarrollo
                                                                    </label>
                                                            </div>
                                                    </div>
                                                    <div class="form-group" ng-
hide="selectedState === 'en desarrollo'">
                                                            <label for="date"
class="col-md-3 control-label">Fecha de presentación</label>
                                                            <div class="col-md-
9">
                                                                    <input
type="text" ng-model="project.date" name="date" class="form-control" date-
field>
                                                                    <div ng-
show="update.$submitted || update.date.$touched">

        <span class="help-block" ng-show="update.date.$error.date">Formato
de fecha inválido (dd/mm/aaaa)</span>

        <span class="help-block" ng-show="update.date.$error.max">La fecha
no puede ser mayor que la actual</span>
                                                                    </div>
                                                            </div>
                                                    </div>
                                                    <div class="form-group" ng-
hide="selectedState === 'en desarrollo'">
```

```
                                                    <label for="mark"
class="col-md-3 control-label">Calificación</label>
                                                    <div class="col-md-
9">
                                                        <input
type="number" ng-model="project.mark" name="mark" min="0" max="10"
class="form-control">
                                                        <div ng-
show="update.$submitted || update.mark.$touched">

        <span class="help-block" ng-show="update.mark.$error.min ||
update.mark.$error.max">La calificación debe tener un valor entre 0 y
10</span>
                                                        </div>
                                                    </div>
                                                </div>
                                                <div class="button-
container">
                                                    <button
type="button" class="btn btn-default" ng-
click="reset(update)">Restablecer</button>
                                                    <button
type="submit" class="btn btn-warning">Actualizar</button>
                                                </div>
                                            </form>
                                            <div ng-show="resultProject"
class="small-container result">
                                                <div ng-show="success"
class="info">
                                                    <div class="alert
alert-success" role="alert">{{ success }}</div>
                                                </div>
                                                <div class="row">
                                                    <hr>
                                                </div>
                                                <div class="row">
                                                    <p><strong>Título:
</strong>{{ resultProject.title }}</p>
                                                    <p><strong>Autor:
</strong>{{ resultProject.studentName }}</p>
                                                    <p><strong>Tutores:
</strong>{{ resultProject.tutor }} <span ng-hide="!resultProject.cotutor ||
resultProject.cotutor.length === 0">y {{ resultProject.cotutor }}</span></p>
                                                    <hr>
```

```
                                                    <p><strong>Estado
del proyecto: </strong>{{ resultProject.state }}</p>
                                                    <p ng-
hide="!resultProject.date || resultProject.date.length === 0"><strong>Fecha
de presentación: </strong>{{ resultProject.date }}</p>
                                                    <p ng-
hide="!resultProject.mark || resultProject.mark.length ===
0"><strong>Calificación: </strong>{{ resultProject.mark }}</p>
                                                </div>
                                            </div>
                                        </div>
                                    </div>
                                </div>
                            </div>
</section>
```

La tercera acción que podemos realizar es **eliminar** y la estructura HTML es idéntica a las dos acciones anteriores, pero con un formulario mucho más simple. Contiene un solo campo de *type="text"* donde introducir la clave del estudiante que se quiere eliminar. En el campo se utiliza el atributo *placeholder* que es un texto de fondo que ayuda al usuario a entender qué información debe introducir en la entrada de texto. Es un atributo que reemplaza y/o refuerza al *label*.

En el formulario solo tenemos un botón de envío y usa la etiqueta *<button type="submit">*.

El bloque con el resultado únicamente incluye un * * con un mensaje de éxito.

```
<section id="modal-delete-container">
        <div class="modal fade" id="modal-delete" role="dialog" aria-
labelledby="modal-delete-label">
                <div class="modal-dialog" role="dialog">
                        <div class="modal-content">
                                <div class="modal-header">
                                        <button type="button"
class="close" data-dismiss="modal" aria-label="Cerrar">
                                                <span aria-
hidden="true">&times;</span>
                                        </button>
                                        <h4 class="modal-title"
id="modal-delete-label">Eliminar proyecto</h4>
                                </div>
```

```
                                <div ng-hide="!error || error.length ===
0" class="modal-error">
                                        <h4 class="modal-title">{{ error
}}</h4>
                                </div>
                                <div class="modal-body">
                                        <form ng-hide="resultDelete"
id="delete" name="delete" ng-submit="submitDeleteForm(delete)"
class="form-horizontal" novalidate>
                                                <div class="form-group">
                                                        <label for="key"
class="col-md-2 control-label">Clave</label>
                                                        <div class="col-
md-10">
                                                                <input
required type="text" ng-model="key" name="key" class="form-control"
placeholder="Clave del estudiante">
                                                                <div ng-
show="delete.$submitted || delete.key.$touched">

        <span class="help-block" ng-
show="delete.key.$error.required">Introduce la clave de
estudiante</span>
                                                                </div>
                                                        </div>
                                                </div>
                                                <div class="button-
container">
                                                        <button
type="submit" class="btn btn-danger">Eliminar</button>
                                                </div>
                                        </form>
                                        <div ng-show="resultDelete"
class="small-container result">
                                                <div ng-show="success"
class="info">
                                                        <div class="alert
alert-success" role="alert">{{ success }}</div>
                                                </div>
                                        </div>
                                </div>
                        </div>
                </div>
        </div>
</section>
```

La última acción, **buscar**, tiene una estructura idéntica a las tres anteriores. El formulario tiene un único campo de texto donde introducir la clave, al igual que en la anterior acción eliminar; y el bloque de resultados tiene tanto un *div* con un mensaje de éxito como otro para mostrar la información devuelta por el servidor, en este caso, texto plano con la información del estudiante que se ha encontrado con la clave introducida:

```
<section id="modal-search-container">
        <div class="modal fade" id="modal-search" role="dialog" aria-
labelledby="modal-search-label">
                <div class="modal-dialog" role="dialog">
                        <div class="modal-content">
                                <div class="modal-header">
                                        <button type="button" class="close"
data-dismiss="modal" aria-label="Cerrar">
                                                <span aria-
hidden="true">&times;</span>
                                        </button>
                                        <h4 class="modal-title" id="modal-
search-label">Buscar proyecto</h4>
                                </div>
                                <div ng-hide="!error || error.length === 0"
class="modal-error">
                                        <h4 class="modal-title">{{ error
}}</h4>
                                </div>
                                <div class="modal-body">
                                        <form ng-hide="resultSearch"
id="search" name="search" ng-submit="submitSearchForm(search)" class="form-
horizontal" novalidate>
                                                <div class="form-group">
                                                        <label for="key"
class="col-md-2 control-label">Clave</label>
                                                        <div class="col-md-
10">
                                                                <input
required type="text" ng-model="key" name="key" class="form-control"
placeholder="Clave del estudiante">
                                                                <div ng-
show="search.$submitted || search.key.$touched">

        <span class="help-block" ng-
show="search.key.$error.required">Introduce la clave de estudiante</span>
                                                                </div>
                                                        </div>
```

```
                                                    </div>
                                                    <div class="button-
container">
                                                        <button
type="submit" class="btn button-search">Buscar</button>
                                                    </div>
                                                </form>
                                                <div ng-show="resultSearch"
class="small-container result">
                                                    <div ng-show="success"
class="info">
                                                        <div class="alert
alert-success" role="alert">{{ success }}</div>
                                                        <p><strong>Detalles
del proyecto: </strong>{{ resultSearch }}</p>
                                                    </div>
                                                </div>
                                            </div>
                                        </div>
                                    </div>
                                </div>
</section>
```

3.5.2. Estilo

Al igual que anteriormente, usamos Bootstrap, el fichero común de estilos *app.css* y unos estilos propios de la página *administracion.css*.

La sección superior de esta página es igual a las demás. Definimos el fondo de la sección con una imagen:

```
#upper-section {
        height: 300px;
        background-image: url(../img/background-administracion.jpg);
        background-size: cover;
}
```

La pestaña informativa tiene un color de fondo acorde con el tema de la página de la administración: el verde.

```
.iconic-information-block {
        margin: 0 auto;
        float: none;
```

```
        background-color: #4cb547;
        color: #fff;
        text-transform: uppercase;
}
```

Cada uno de los botones de acción ocuparán un cuarto del tamaño del padre en pantallas mayores de 992px gracias a las clase de Bootstrap que permite organizar un *grid*: en este caso si el botón ocupará un cuarto, habiendo doce columnas en el *grid,* usamos la clase *col-md-3*. Para pantallas con un ancho mínimo de 768px usamos la clase *col-sm-6* con lo que cada botón ocupará la mitad del ancho del padre, es decir, habrá dos botones por fila:

```
<div class="action-container col-md-3 col-sm-6">
```

Al contenedor de las acciones le damos una separación superior con *margin-top* y centramos su contenido con *text-align: center*. Al *div* que contiene las propias acciones lo separamos del texto descriptivo al principio del contenedor con un margen superior. También separamos por debajo los contenedores de cada acción con un *margin-bottom* y lo centramos horizontalmente con *margin: 0 auto*:

```
#actions-container {
        margin-top: 40px;
}

#actions-header-container {
        text-align: center;
}

#actions {
        margin-top: 25px;
}

.action-container {
        margin: 0 auto;
        margin-bottom: 20px;
}
```

Para conseguir que el *div* tenga aspecto de botón redondeado le aplicamos *border-radius: 50%*:

```
#actions .iconic-information-block {
```

```
        border-radius: 50%;
}
```

Finalmente, centramos el contenido del botón, que es un icono usando la etiqueta <*span*> con las clases de iconos de Bootstrap *glyphicon* y, abajo, un párrafo con la descripción de la acción. Para lograr centrar este contenido usamos los nuevos atributos de CSS3 *display: flex* y *justify-content: center* además del usual *margin: 0 auto*. Sobre el contenedor interno del contenido del botón aplicamos la propiedad también CSS3 *align-self: center* para alinear el contenido. Finalmente, añadimos la propiedad *cursor: pointer* sobre el contenedor exterior para cambiar el aspecto de cursor al pasarlo sobre el bloque y comunique al usuario que es un botón que puede pulsar:

```
.action {
        display: flex;
        justify-content: center;
        width: 150px;
        height: 150px;
        margin: 0 auto;
        cursor: pointer;
}

.action-inner-container {
        align-self: center;
}
```

Para cada una de las acciones asignamos un color de fondo diferente que las haga distinguibles:

```
.action.create {
        background-color: #79dede;
}

.action.update {
        background-color: #dec461;
}

.action.delete {
        background-color: #de7872;
}

.action.search {
        background-color: #daaae4;
```

```
}
```

Con esto se han aplicado estilos a todos los elementos que aparecen por primera vez al acceder a la página de administración. Ahora pasemos a aplicar los estilos a los cuatro *modals* con los formularios de acción.

Primero, permitimos desplazamiento vertical con *overflow-y: scroll* en el caso de que el alto del *modal* sea mayor al de la pantalla:

```
#main-section .modal {
        overflow-y: scroll;
}
```

Aplicamos un *media query* que modifique el ancho del *modal* a 800px para que se ajuste más al ancho de la pantalla para resoluciones más bajas ya que por defecto Bootstrap lo hace demasiado estrecho y no se ve correctamente el formulario:

```
@media (min-width: 768px) {
        #modal-create .modal-dialog, #modal-update .modal-dialog {
                width: 800px;
        }
}
```

Acorde con el tema verde de la página cambiamos el fondo de la cabecera del *modal:*

```
.modal-header {
        background-color: #4cb547;
        color: #fff;
}
```

Posicionamos el bloque de error sobre la cabecera del *modal* usando la propiedad *position: absolute* e indicamos la posición con *top: 0* y *left: 0* lo que quiere decir que el bloque se posicionará justo arriba a la izquierda del padre. El ancho será el total del padre y tendrá un fondo rojo representando error:

```
.modal-error {
        position: absolute;
        top: 0;
        left: 0;
        padding: 15px;
```

```
        background-color: #de423b;
        color: #fff;
        width: 100%;
}

.modal-error span {
        text-align: center;
}
```

Cambiamos el aspecto por defecto del botón de cierre del *modal*. Con la pseudoclase *hover* podemos aplicar estilo al elemento cuando se pase el cursor sobre él:

```
.close {
        color: #fff;
        opacity: 0.8;
}

.close:hover {
        color: #fff;
        opacity: 0.5;
}
```

Añadimos *padding* al cuerpo del *modal* para que el contenido (formulario) no quede tan pegado a los límites:

```
.modal-body {
        padding: 25px 60px;
}
```

Centramos los botones, que por defecto con Bootstrap se alinean a la izquierda, y aumentamos el tamaño que viene de la clase *btn*:

```
.button-container {
        text-align: center;
}

.button-container button {
        margin: 5px;
        min-width: 110px;
}
```

Para los botones de envío de crear, actualizar y borrar hemos utilizado clases contextuales de Bootstrap para botones: *btn-info, btn-warning* y *btn-danger*. Estas clases aplican un color de letra, de fondo, y de borde a los botones, así como para el estado *hover*. Para el botón de buscar, no hemos usado una de las clases contextuales de Bootstrap ya que hemos elegido una gama de colores rosa y el *framework* no tiene una clase similar que encaje. Así que añadimos estilos para el botón buscar:

```
.button-search {
        background-color: #daaae4;
        border-color: #c293c9;
        color: #fff;
}

.button-search:hover {
        background-color: #c293c9;
        color: #fff;
}
```

Las clases usadas a continuación son autogeneradas por AngularJS al validar el formulario (la validación de formularios con AngularJS se explicará en una sección más adelante). Aplicamos estilos a las mismas, para dar un aspecto de error a los campos de entrada donde la validación haya fallado:

```
form.ng-submitted .ng-valid, .ng-valid.ng-touched {
        border-color: #3c763d;
        -webkit-box-shadow: inset 0 1px 1px rgba(0,0,0,.075);
        box-shadow: inset 0 1px 1px rgba(0,0,0,.075);
}

form.ng-submitted .ng-valid:focus, .ng-valid.ng-touched:focus {
        border-color: #2b542c;
        -webkit-box-shadow: inset 0 1px 1px rgba(0,0,0,.075),0 0 6px
#67b168;
        box-shadow: inset 0 1px 1px rgba(0,0,0,.075),0 0 6px #67b168;
}

form.ng-submitted .ng-invalid, .ng-invalid.ng-touched {
        border-color: #a94442;
        -webkit-box-shadow: inset 0 1px 1px rgba(0,0,0,.075);
        box-shadow: inset 0 1px 1px rgba(0,0,0,.075);
}

form.ng-submitted .ng-invalid:focus, .ng-invalid.ng-touched:focus {
```

```
        border-color: #843534;
        -webkit-box-shadow: inset 0 1px 1px rgba(0,0,0,.075),0 0 6px
#ce8483;
        box-shadow: inset 0 1px 1px rgba(0,0,0,.075),0 0 6px #ce8483;
}
```

La clase *help-block* se utiliza sobre los mensajes de error que aparecen debajo de cada campo del formulario cuya validación haya fallados y le aplica un color rojizo:

```
.help-block {
        color: #a94442;
}
```

El HTML con los mensajes de error se incluirán en el apartado de la validación con Angular.

Por último, ajustamos los mensajes de éxito que aparecen después de enviar el formulario con respuesta positiva del servicio:

```
.info {
        text-align: center;
        margin: 0 auto;
}

.info .alert {
        width: 100%;
}

.result {
        margin: 0 auto;
        text-align: center;
}
```

En la Figura 5 podemos ver cómo se ve la página de administración con los estilos:

Figura 5. Página de administración

3.5.3. JavaScript

Esta página utiliza mucho código JavaScript, tanto de Bootstrap como AngularJS, pero fundamentalmente Angular.

Incluimos el fichero *js* de Bootstrap en *<head>*:

```
<script src="js/bootstrap.js"></script>
```

La función JavaScript de Bootstrap que utilizamos en esta página es mostrar los *modals*. Al hacer clic sobre uno de los botones de acción debe haber un mecanismo que desencadene la acción de mostrar el *modal* y una forma de indicar **qué *modal*** se ha de mostrar. Para ello, Bootstrap, al igual que Angular, utiliza atributos HTML propios.

Veamos, por ejemplo, el botón "crear"; todos los demás funcionan del mismo modo. El atributo usado por Bootstrap para indicar que al hacer clic sobre el elemento se debe desencadenar un comportamiento *modal* es *data-toggle="modal"*. A continuación, se indica el elemento sobre el que se ha de desencadenar ese comportamiento con el atributo *data-target="#modal-create"* lo que quiere decir el elemento con la *id="modal-create"*:

```
<div data-toggle="modal" data-target="#modal-create">
```

El contenedor del *modal* ha de ser identificado de forma acorde:

```
<div class="modal fade" id="modal-create" role="dialog" aria-
labelledby="modal-create-label">
```

Este uso de JavaScript con Bootstrap y *modals* se da para las otras tres acciones.

AngularJS se usa tanto para el envío de los formularios como para su validación. En este apartado veremos el envío, mientras que dedicaremos un apartado especial más adelante para explicar la validación.

Con Angular podemos definir qué queremos que ocurra cuando se envíe un formulario con la directiva *ng-subimit* y el valor de la función declarado que debe ejecutarse cuando esto ocurra. Veamos el formulario **crear**:

```
<form ng-hide="resultProject" id="create" name="create" ng-
submit="submitCreateForm(create)" class="form-horizontal" novalidate>
```

Esto quiere decir que cuando se envíe el formulario, si pasa la validación, se ejecutará la función *submitCreateForm(create)*. El argumento *create* es necesario y corresponde con el nombre (*name*) del formulario que queremos tratar en nuestro código.

Antes de declarar el método, veamos la directiva *ng-model*. Por ejemplo, en el campo de entrada del nombre:

```
<input required type="text" ng-model="project.name" ng-pattern="/^[a-z\u00E0-
\u00FC ]+$/i" name="name" class="form-control">
```

La directiva *ng-model* crea un vínculo entre el valor del campo y una variable declarada como *project.name*. Esta variable será tratada como un modelo, con lo que, entre otras cosas, podrá ser validada y será accesible desde nuestra aplicación Angular.

El siguiente paso es dirigirnos a nuestro fichero *app.js* y añadimos el controlador de la página que, recordamos, definimos sobre la etiqueta *body* con *ng-controller="AdminCtrl"*:

```
<body ng-controller="AdminCtrl">
```

Declaramos el cuerpo del controlador:

```
app.controller('AdminCtrl', function($scope, ws) {
```

```
});
```

Y, a continuación, el método *submitCreateForm:*

```
$scope.submitCreateForm = function(form) {
        var project = $scope.project;
        var params = {
                "nombre": project.name,
                "primerApellido": project.firstLastName,
                "segundoApellido": project.secondLastName,
                "tituloProyecto": project.title,
                "tutor1": project.tutor,
                "tutor2": project.cotutor,
                "estadoProyecto": project.state,
                "fechaPresentacionProyecto": project.date ?
dateFormat(project.date, "dd-mm-yyyy") : "",
                "calificacionProyecto": project.mark
        };

        if (form.$valid) {
                ws.postCreateStudent(params).then(function(data) {
                        if (!data || data.length === 0) {
                                $scope.error = "Ha habido un problema en el
servidor";
                        } else {
                                $scope.success = "Se ha creado el proyecto
con éxito";
                                $scope.error = null;
                                $scope.reset($scope.create);

                                $scope.resultProject = {};
                                $scope.resultProject = {
                                        "studentName": (data.nombre + " " +
data.primerApellido + " " + data.segundoApellido).trim(),
                                        "title": data.tituloProyecto,
                                        "tutor": data.tutor1,
                                        "cotutor": data.tutor2,
                                        "state":
data.estadoProyecto.charAt(0).toUpperCase() + data.estadoProyecto.slice(1),
                                        "date":
data.fechaPresentacionProyecto,
                                        "mark": data.calificacionProyecto
                                };
```

```
                        }
                });
        }
};
```

Este método se ejecutará cuando se envíe el formulario de crear proyecto. Primero, crea un *hash* con los parámetros enviados por el formulario, pero solamente aquellos que piden el servicio web para crear un nuevo proyecto.

A continuación, si la validación del formulario pasa, llamamos al método del *factory* que invoca al servicio web para crear un proyecto. Si la respuesta es nula o vacía, quiere decir que ha habido un problema en el servidor y se mostrará un mensaje de error sobre la cabecera del *modal*. Recordamos el HTML:

```
<div ng-hide="!error || error.length === 0" class="modal-error">
        <h4 class="modal-title">{{ error }}</h4>
</div>
```

La directiva *ng-hide* la hemos visto anteriormente y hará que el *div* de error se oculte si no hay un error. De lo contrario se muestra el error contenido en la variable *error* que, recordamos, se ha asignado a *$scope.error*.

Si hay respuesta no nula desde el servidor esperamos un JSON con los datos del proyecto o estudiante creado. Entonces establecemos un mensaje de éxito en *$scope.success*, restablecemos el valor de *$scope.error* por si hubo anteriormente un error y formamos un *hash* con los datos del proyecto, que se mostrarán junto con el mensaje de éxito:

```
<div ng-show="resultProject" class="small-container result">
        <div ng-show="success" class="info">
                <div class="alert alert-success" role="alert">{{ success
}}</div>
        </div>
        <div class="row">
                <hr>
        </div>
        <div class="row">
                <p><strong>Título: </strong>{{ resultProject.title }}</p>
                <p><strong>Autor: </strong>{{ resultProject.studentName
}}</p>
```

```
                <p><strong>Tutores: </strong>{{ resultProject.tutor }} <span
ng-hide="!resultProject.cotutor || resultProject.cotutor.length === 0">y {{
resultProject.cotutor }}</span></p>
                <hr>
                <p><strong>Estado del proyecto: </strong>{{
resultProject.state }}</p>
                <p ng-hide="!resultProject.date || resultProject.date.length
=== 0"><strong>Fecha de presentación: </strong>{{ resultProject.date }}</p>
                <p ng-hide="!resultProject.mark || resultProject.mark.length
=== 0"><strong>Calificación: </strong>{{ resultProject.mark }}</p>
        </div>
</div>
```

La función usada para invocar el método del servicio que crea un estudiante se añade a nuestro *factory* anteriormente definido:

```
return {
        getAllStudents: function() {
                // ...
        },
        postCreateStudent: function(json) {
                var url = full_path + create_estudiante_path;
                var data = JSON.stringify(json);
                return $http.post(url, data).then(function(response) {
                        return response.data;
                });
        },
}
```

El contenedor externo *.result* es hermano del formulario. Podemos ver la directiva *ng-show=”resultProject”*, en contraposición a *ng-hide=”resultProject”* del formulario*:*

```
<form ng-hide="resultProject" id="create" name="create" ng-
submit="submitCreateForm(create)" class="form-horizontal" novalidate>
```

Como es de esperar, *ng-show* hace lo contrario de *ng-hide*: muestra el elemento si se cumple la expresión indicada. Así, conseguimos ocultar el formulario y mostrar los

resultados si hay un proyecto resultado (de la creación), y lo contrario si no hay ningún *resultProject* no nulo.

Además del envío del formulario, tenemos el botón de restaurar el estado del formulario. El comportamiento de la acción de hacer clic sobre un botón se indica con la directiva *ng-click:*

```
<button type="button" class="btn btn-default" ng-
click="reset(create)">Restablecer</button>
```

Declaramos el método en el controlador:

```
$scope.reset = function(form) {
        $scope.project = {
                name: "",
                firstLastName: "",
                secondLastName: "",
                email: "",
                title: "",
                topic: "Domótica",
                tutor: "",
                cotutor: "",
                state: "presentado",
                date: "",
                mark: ""
        };
        $scope.selectedState = $scope.project.state;
        form.$setPristine();
        form.$setUntouched();
}
```

En esta función restauramos los valores del proyecto para que se muestren los campos vacíos y usamos los métodos de *form $setPristine()* y *$setUntouched()* para eliminar las clases autogeneradas aplicadas sobre el formulario y los campos que indican que hemos colocado el cursor sobre ellos y que el formulario y los campos han sido validados.

Con esto queda explicado el código de AngularJS para el formulario crear. A continuación, presentamos la acción **actualizar**, que no explicaremos con detalles pues su funcionamiento es idéntico al de la acción crear.

En este caso el método llamado al enviar el formulario es *submitUpdateForm:*

```
<form ng-hide="resultProject" id="update" name="update" ng-
submit="submitUpdateForm(update)" class="form-horizontal" novalidate>
```

Y las diferencias en el formulario, además de tener menos campo, es que añadimos un campo de texto con la clave del proyecto que queremos actualizar e identificamos con el *ng-model* de *project.key:*

```
<input required type="text" ng-model="project.key" name="key" class="form-
control" placeholder="Clave del estudiante">
```

El botón de restaurar formulario hace uso del mismo método que crear, pero en este caso pasamos como argumento el formulario de actualizar, que tiene *name=”update”:*

```
<button type="button" class="btn btn-default" ng-
click="reset(update)">Restablecer</button>
```

La lógica del método que se ejecuta al enviar el formulario de actualización es similar al método de crear:

```
$scope.submitUpdateForm = function(form) {
        var project = $scope.project;
        var key = project.key;
        var params = {
                "tituloProyecto": project.title,
                "estadoProyecto": project.state,
                "fechaPresentacionProyecto": project.date ?
dateFormat(project.date, "dd-mm-yyyy") : "",
                "calificacionProyecto": project.mark
        };

        if (form.$valid) {
                ws.putUpdateStudent(key, params).then(function(data) {
                        if (!data || data.length === 0) {
                                $scope.error = "No existe un estudiante con
esa clave";
```

```
                        } else {
                                $scope.success = "Se ha actualizado el
proyecto con éxito";
                                $scope.error = null;
                                $scope.reset($scope.update);

                                $scope.resultProject = {};
                                $scope.resultProject = {
                                        "studentName": (data.nombre + " " +
data.primerApellido + " " + data.segundoApellido).trim(),
                                        "title": data.tituloProyecto,
                                        "tutor": data.tutor1,
                                        "cotutor": data.tutor2,
                                        "state":
data.estadoProyecto.charAt(0).toUpperCase() + data.estadoProyecto.slice(1),
                                        "date":
data.fechaPresentacionProyecto,
                                        "mark": data.calificacionProyecto
                                };
                        }
                });
        }
}
```

Y el método del servicio *putUpdateStudent(key, params)*:

```
return {
        getAllStudents: function() {
                // ...
        },
        postCreateStudent: function(json) {
                // ...
        },
        putUpdateStudent: function(key, json) {
                var url = full_path + update_estudiante_path + "/" + key;
                var data = JSON.stringify(json);
                return $http.put(url, data).then(function(response) {
                        return response.data;
                });
        }
}
```

La siguiente acción, **borrar**, sigue el mismo mecanismo de Angular que las acciones anteriores.

El método a ejecutar al enviar el formulario es *submitDeleteForm:*

```
<form ng-hide="resultDelete" id="delete" name="delete" ng-
submit="submitDeleteForm(delete)" class="form-horizontal" novalidate>
```

Dicho método se declara como sigue:

```
$scope.submitDeleteForm = function(form) {
        if (form.$valid) {
                ws.deleteStudent($scope.key).then(function(data) {
                        if (data === true) {
                                $scope.resultDelete = true;
                                $scope.success = "Se ha eliminado el
proyecto con éxito";

                                $scope.error = null;
                        } else {
                                $scope.error = "No existe un estudiante
con esa clave";
                        }
                });
        }
}
```

Al enviar el formulario simplemente se establece un mensaje de error que se mostrará en la cabecera si el estudiante no existe, o bien se genera un mensaje de éxito mostrado en el bloque de resultado.

La variable *resultDelete* es la usada por *ng-hide* y *ng-show* para mostrar y/o ocultar el formulario y/o *div* de resultado según la respuesta del servicio.

El método del *factory* hace una petición DELETE al servicio web para eliminar un proyecto de la base de datos:

```
return {
        getAllStudents: function() {
                // ...
        },
        postCreateStudent: function(json) {
                // ...
        },
```

```
        putUpdateStudent: function(key, json) {
                // ...
        },
        deleteStudent: function(key) {
                var url = full_path + delete_estudiante_path + "/" + key;
                return $http.delete(url).then(function(response) {
                        return response.data;
                });
        }
}
```

Por último, veamos la acción **buscar**. Su funcionamiento es similar a las acciones anteriores.

El método ejecutado al enviar el formulario es *submitSearchForm:*

```
<form ng-hide="resultSearch" id="search" name="search" ng-
submit="submitSearchForm(search)" class="form-horizontal" novalidate>
```

Y se define como sigue:

```
$scope.submitSearchForm = function(form) {
        if (form.$valid) {
                ws.getSearchStudent($scope.key).then(function(data) {
                        if (!data || data.length === 0) {
                                $scope.error = "No existe un estudiante con
esa clave";
                        } else {
                                $scope.resultSearch = data;
                                $scope.success = "Se ha encontrado el
proyecto";

                                $scope.error = null;
                        }
                });
        }
}
```

La lógica es similar a la acción borrar. Si el servicio no encuentra un estudiante con esa clave, se establece un mensaje de error en *$scope.error* que se mostrará en la cabecera del formulario. De lo contrario se mostrará un mensaje de éxito en *$scope.success* que se mostrará en el bloque de resultados, así como una variable

$scope.resultSearch con un texto plano devuelto por el servicio que contiene la información del proyecto. Esta variable servirá tanto para mostrar la información del proyecto como para su uso en las directivas *ng-show* y *ng-hide* con el mismo propósito que en anteriores ocasiones.

El método del servicio *factory* que hace una petición al servicio para encontrar un estudiante:

```
return {
        getAllStudents: function() {
                // ...
        },
        postCreateStudent: function(json) {
                // ...
        },
        putUpdateStudent: function(key, json) {
                // ...
        },
        deleteStudent: function(key) {
                // ...
        },
        getSearchStudent: function(key) {
                        var url = full_path + get_estudiante_path + "/" +
key;
                        return $http.get(url).then(function(response) {
                                return response.data;
                        });
                }
}
```

Con esto queda explicado todo el comportamiento de AngularJS en nuestros formularios en cuanto a envío, restablecimiento de valores y mostrar y ocultar elementos del DOM.

En la siguiente sección veremos detalladamente el uso de Angular para validar los formularios.

4. Validación de formularios

Para la validación de los formularios utilizamos AngularJS, pues proporciona un sistema de validación de campos de entrada para validar los campos HTML5 más comunes, como *text, number, url, email, date, radio* o *checkbox* y una serie de directivas que aplican la validación sobre ese campo, como *required, pattern, minlength, maxlength, min* o *max.* Para otro tipo de validaciones definimos directivas personalizadas.

Como vimos anteriormente, Angular al validar un formulario aplica una serie de estilos al formulario, así como al campo que ha sido validado. En el punto de estilos de la página de administración vimos cómo aplicamos los diferentes estilos a dichos elementos.

Para que un formulario sea validado basta con usar cualquiera de los *input* mencionados y aplicar la directiva que aplique la validación. Es también de vital importancia definir un modelo sobre el campo usando la directiva *ng-model*. La validación se aplica sobre los modelos incluidos en un formulario.

Antes de pasar a explicar cada uno de los formularios, hemos de comentar que deshabilitaremos la validación de campos nativa del navegador, función nueva de HTML5, para evitar información redundante e incompatibilidad, ya que tanto la validación como la presentación de mensajes de error para cada campo las implementaremos con Angular. Para deshabilitar la validación nativa del navegador utilizamos el atributo *novalidate* sobre el formulario <*form*>, por ejemplo, para el formulario crear:

```
<form ng-hide="resultProject" id="create" name="create" ng-
submit="submitCreateForm(create)" class="form-horizontal" novalidate>
```

Recordamos que en las secciones donde se explicaba la estructura HTML, en el caso de la página de administración (donde aplicaremos la validación de formularios), omitimos el bloque donde se incluían los mensajes de error de la validación, pues dedicaremos esta sección para su explicación.

En todos los casos, este bloque de código HTML donde se incluyen los mensajes de error, se trata de un <*div*> hermano del campo de entrada (*input*), es decir, si el campo de entrada es el primer hijo del elemento padre, el bloque de validación es el

segundo hijo. Veamos, por ejemplo, la estructura HTML completa del campo "nombre" en el formulario crear (el *div.form-group*) para comprender lo explicado:

```
<div class="form-group">
        <label for="name" class="col-md-3 control-label">Nombre</label>
        <div class="col-md-9">
                <input required type="text" ng-model="project.name" ng-
pattern="/^[a-z\u00E0-\u00FC ]+$/i" name="name" class="form-control">
                <div ng-show="create.$submitted || create.name.$touched">
                        <span class="help-block" ng-
show="create.name.$error.required">El nombre es obligatorio</span>
                        <span class="help-block" ng-
show="create.name.$error.pattern">Caracteres inválidos</span>
                </div>
        </div>
</div>
```

El propósito de esta aclaración es evitar copiar el código completo de todo el *div.form-group* y contenedores internos a la hora de explicar la validación, donde solamente incluiremos el *input* y el *div* de validación.

Por último, cabe mencionar que un campo se puede validar por Angular en dos momentos diferentes:

- Cuando una vez hagamos clic dentro de un campo (y añadamos o no información), salgamos de él (perder el *focus*) haciendo clic en otro campo o en otro lugar del formulario. Esto desencadenará la validación del dato introducido en ese campo y se aplicarán las pertinentes clases automáticamente
- Cuando hagamos clic sobre el botón de enviar formulario o lo enviemos de algún otro modo. Esto hará que se validen todos los campos del formulario.

4.1. Formulario crear

Veamos en primer lugar un vista general del nuestro formulario para crear un nuevo proyecto (Figura 6). Como explicamos en la estructura del HTML, éste se encuentra en un *modal* de Bootstrap que se muestra por medio de JavaScript cuando hagamos clic sobre el botón "crear".

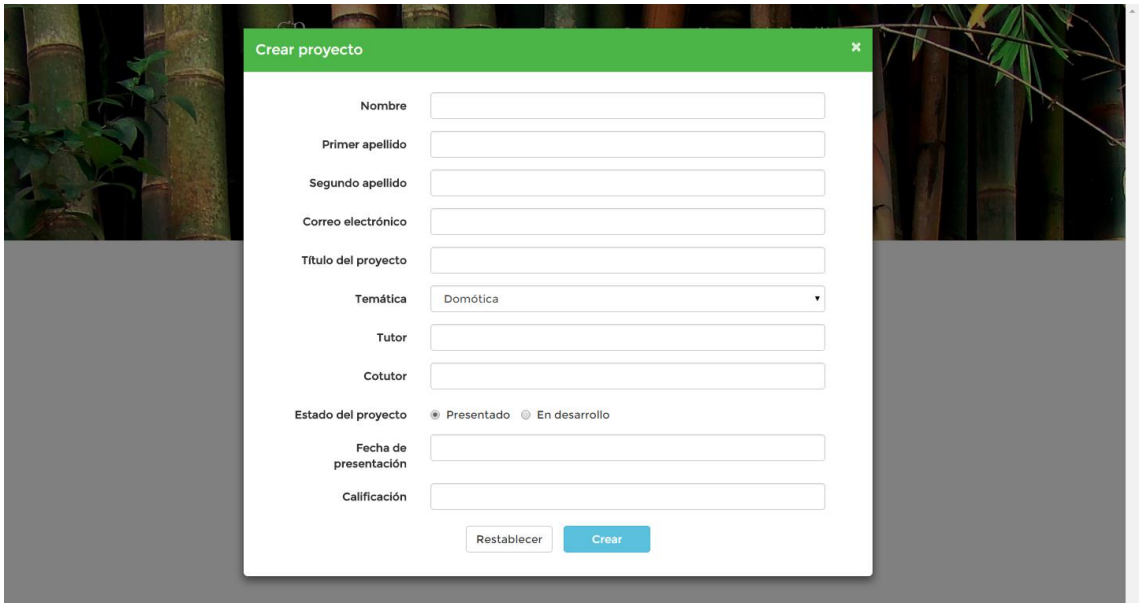

Figura 6. Formulario crear

El primer campo de este formulario es el **nombre** del estudiante:

```
<input required type="text" ng-model="project.name" ng-pattern="/^[a-z\u00E0-\u00FC ]+$/i" name="name" class="form-control">
```

Como explicamos en la introducción de la sección, ligamos este campo a un *ngModel* haciendo uso de la directiva *ng-model* con la variable *project.name*. Esto logrará que el campo sea validado automáticamente:

```
        <input required type="text" ng-model="project.name" ng-pattern="/^[a-z\u00E0-\u00FC ]+$/i" name="name" class="form-control">
```

El atributo *required* es un atributo HTML5 aprovechado por Angular para aplicar la una validación que comprueba si se ha introducido un dato o no. Indica que el campo es obligatorio.

La directiva *ng-pattern* aplicará una validación de formato: validará que el dato introducido concuerda con la expresión regular indicada. En este caso, que el nombre se forme por letras, incluidas aquéllas con acentos.

Como comentamos, a continuación del campo de entrada tenemos un *div* que muestra mensajes de error, si hubiera:

```
<div ng-show="create.$submitted || create.name.$touched">
        <span class="help-block" ng-show="create.name.$error.required">El
nombre es obligatorio</span>
        <span class="help-block" ng-
show="create.name.$error.pattern">Caracteres inválidos</span>
</div>
```

Primero, haciendo uso de *ng-show* mostramos el *div* que contiene los mensajes de error si el formulario ha sido enviado (*créate.$submitted*) o si el campo nombre ha sido tocado (*create.name.$touched*).

Mostraremos un mensaje de error usando de nuevo *ng-show* diciendo que el campo es obligatorio si no se ha introducido ningún dato. Para hacer esta comprobación usamos *create.name.$error.required*.

El objeto *formulario.campo.$error* contiene un *hash* de información con los distintos errores del formulario.

El siguiente mensaje se mostrará si el nombre introducido no usa los caracteres alfabéticos permitidos usando el valor devuelto por *create.name.$error.pattern*.

Los campos que vienen a continuación son los de **primer apellido** y **segundo apellido**. En el campo de primer apellido aplicamos la misma validación que para el campo nombre: es un campo requerido y debe seguir el patrón expresado:

```
<input required type="text" ng-model="project.firstLastName" ng-
pattern="/^[a-z\u00E0-\u00FC ]+$/i" name="firstLastName" class="form-
control">
```

De forma análoga al nombre, muestra los mensajes de error pertinentes:

```
<div ng-show="create.$submitted || create.firstLastName.$touched">
        <span class="help-block" ng-
show="create.firstLastName.$error.required">El primer apellido es
obligatorio</span>
```

```
        <span class="help-block" ng-
show="create.firstLastName.$error.pattern">Caracteres inválidos</span>
</div>
```

El campo segundo apellido no es obligatorio, tal y como implementa el servicio web, porque puede haber un estudiante que no tenga segundo apellido. Por tanto, no incluye el atributo *required*. No obstante, si se introdujese un segundo apellido, se validaría su formato:

```
<input type="text" ng-model="project.secondLastName" ng-pattern="/^[a-
z\u00E0-\u00FC ]+$/i" name="secondLastName" class="form-control">
```

Habría un único mensaje de error a mostrar posible, en el caso de que el segundo apellido contuviese caracteres no permitidos:

```
<div ng-show="create.$submitted || create.secondLastName.$touched">
        <span class="help-block" ng-
show="create.secondLastName.$error.pattern">Caracteres inválidos</span>
</div>
```

En la Figura 7 vemos un ejemplo de validación fallida para los campos mencionados:

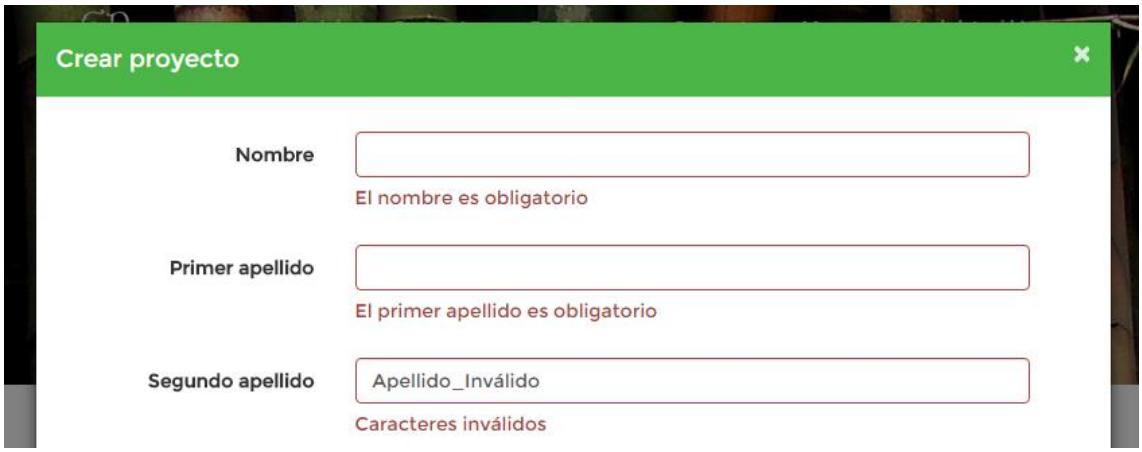

Figura 7. Validación de nombre, primer y segundo apellido

El siguiente campo es el **email**. Al ser un *input* de *type=”email”* Angular validará que el formato del dato introducido es un formato de email válido. Es también un campo requerido al incluir el atributo *required* y, además, es el primer caso donde

utilizamos una validación personalizada. Usamos la directiva *email-uca* que declararemos en nuestro fichero con la lógica de Angular. Esta directiva validará que la entrada del usuario se ajusta al formato de correo que termina en *@alumnos.uca.es*:

```
<input required type="email" ng-model="project.email" name="email"
class="form-control" email-uca>
```

Veamos a continuación la declaración de la directiva:

```
app.directive('emailUca', function() {
        return {
                require: 'ngModel',
                link: function(scope, elm, attrs, ctrl) {
                        ctrl.$validators.emailUca = function(modelValue,
viewValue) {
                                if (ctrl.$isEmpty(modelValue)) {
                                        return true;
                                }

                                var pos = viewValue.indexOf("@alum.uca.es");
                                if (pos > -1 && pos == viewValue.length -
"@alum.uca.es".length) {
                                        return true;
                                }

                                return false;
                        };
                }
        };
});
```

Hemos añadido la declaración de la directiva a continuación del último controlador: *AdminCtrl*. Como podemos observar en el código, añade una nueva regla de validación al objeto *ctrl.$validators*, llamada *emailUca*. Esta función simplemente comprueba que el final del dato incluye la cadena *@alum.uca.es.*

Probablemente se haya podido observar que el nombre de la directiva difiere en su declaración (*emailUca*) de su uso en la etiqueta HTML (*email-uca*). Angular transforma convenientemente la variable con notación *dash* en la variable con notación *camelCase*. Esto es una ventaja ya que es convención usar la notación *dash* en HTML y la notación *camelCase* en JavaScript.

A continuación vemos el *div* con los mensajes de error, que funcionan de igual manera que explicamos en los campos anteriores:

```
<div ng-show="create.$submitted || create.email.$touched">
        <span class="help-block" ng-show="create.email.$error.required">El
correo electrónico es obligatorio</span>
        <span class="help-block" ng-show="create.email.$error.email">Formato
inválido de dirección de correo electrónico</span>
        <span class="help-block" ng-show="create.email.$error.emailUca">La
dirección de correo debe ser del dominio <em>@alum.uca.es</em></span>
</div>
```

En la Figura 8 podemos ver un ejemplo de la validación del email:

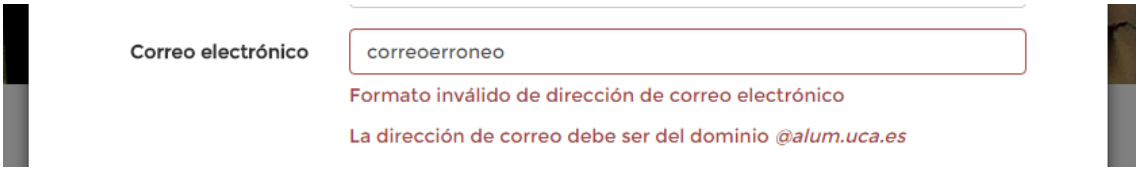

Figura 8. Email con formato incorrecto

Y en la Figura 9 el caso en el que el email tenga un formato correcto, pero no sea *@alumnos.uca.es:*

Figura 9. Email con formato no UCA

El siguiente campo es el de **título del proyecto** y su única validación es que sea obligatorio. Permitimos el uso de cualquier carácter en este caso:

```
<input required type="text" ng-model="project.title" name="title"
class="form-control">
```

El *div* con el mensaje de validación:

```
<div ng-show="create.$submitted || create.title.$touched">
        <span class="help-block" ng-show="create.title.$error.required">El
título del proyecto es obligatorio</span>
```

```
</div>
```

A continuación del título de proyecto se encuentra la **temática** o categoría del proyecto. Este campo no pasa ninguna validación:

```
<select ng-model="project.topic" name="topic" class="form-control">
        <option value="Domótica">Domótica</option>
        <option value="Salud">Salud</option>
        <option value="Turismo">Turismo</option>
</select>
```

Puede parecer extraño que aunque no se utilice el atributo HTML *checked* para el primer valor, "Domótica", aparezca seleccionado éste por defecto al cargar el formulario. La razón es porque hemos ligado el campo con *ng-model*, como hacemos con todos los campos, y hemos inicializado la variable *project* al principio del controlador de administración, *AdminCtrl*. Esto hará que aparezca seleccionada esa opción de forma automática:

```
$scope.project = {
        state: "presentado",
        topic: "Domótica"
};
```

Los siguientes dos campos, **tutor** y **cotutor** siguen una validación igual al nombre y apellidos. El tutor es obligatorio pero el cotutor no, por eso para el primero utilizamos el atributo *required* pero no así para el segundo; no obstante, para los dos validamos su formato con la directiva *ng-pattern*:

```
<input required type="text" ng-model="project.tutor" ng-pattern="/^[a-
z\u00E0-\u00FC ]+$/i" name="tutor" class="form-control">

<input type="text" ng-model="project.cotutor" ng-pattern="/^[a-z\u00E0-\u00FC
]+$/i" name="cotutor" class="form-control">
```

La validación del tutor:

```
<div ng-show="create.$submitted || create.tutor.$touched">
        <span class="help-block" ng-show="create.tutor.$error.required">El
tutor es obligatorio</span>
        <span class="help-block" ng-
show="create.tutor.$error.pattern">Caracteres inválidos</span>
```

```
</div>
```

Y la del cotutor:

```
<div ng-show="create.$submitted || create.cotutor.$touched">
        <span class="help-block" ng-
show="create.cotutor.$error.pattern">Caracteres inválidos</span>
</div>
```

El primero de los tres últimos campos del formulario es el **estado del proyecto**. Este campo no es validado pero utiliza la directiva *ng-change*:

```
<input type="radio" name="state" value="presentado" ng-model="project.state"
ng-change="changeState(project.state)">

<input type="radio" name="state" value="en desarrollo" ng-
model="project.state" ng-change="changeState(project.state)" ng-
click="clearPresentedAttributes()">
```

Esta directiva indica que cuando se cambie el valor del *ngModel* ligado (*project.state*), es decir, cuando se haga clic sobre un *radio* o el otro, se ejecute la función indicada: *changeState(project.state).*

Esta función se define en *AdminCtrl* como sigue:

```
$scope.changeState = function(state) {
        $scope.selectedState = state;
};
```

La función está ligada al *$scope* al igual que las demás que se usan en la vista, para que puedan ser invocadas desde esta ubicación, y su propósito es muy simple, cambia el valor de la variable *selectedState* al valor que se pase como argumento.

Este *selectedState* se utilizará en conjunción con la directiva *ng-hide* sobre los *divs* padre de los campos de calificación y de fecha de presentación de proyecto:

```
<div class="form-group" ng-hide="selectedState === 'en desarrollo'">
```

En definitiva, el objetivo de esto es ocultar los campos de calificación y fecha de presentación si el usuario selecciona como estado del proyecto el valor “en desarrollo”,

pues un proyecto en desarrollo no puede tener una calificación y fecha de presentación asociadas. Si se selecciona de nuevo la opción "presentado", como es de esperar de la directiva *ng-hide*, estos dos campos volverán a mostrarse. El funcionamiento de esto puede verse en la Figura 10.

Aunque el servicio web ignorará el envío de una fecha y una calificación en el caso de que el usuario seleccione un proyecto como "en desarrollo", es buena práctica evitar enviar datos innecesarios, además de ahorrarle tiempo al usuario y evitar su confusión a la hora de rellenar el formulario.

Figura 10. Campos fecha y calificación ocultos al elegir proyecto en desarrollo

El siguiente campo es el de **fecha de presentación del proyecto** y no es obligatorio, bien porque el proyecto está en desarrollo o bien porque la fecha de presentación se desconoce. No obstante, se valida que la fecha elegida no sea mayor que la fecha actual, con la directiva personalizada *date-field:*

```
<input type="text" ng-model="project.date" name="date" class="form-control"
date-field>
```

La directiva la declaramos de igual forma que hicimos la validación del email con formato de la UCA. En *app*.js, a continuación de la directiva *emailUca*, añadimos:

```
app.directive('dateField', function() {
        return {
                require: 'ngModel',
                link: function(scope, elm, attrs, ctrl) {
                        elm.datepicker({
                                        dateFormat: 'dd/mm/yy',
                                        firstDay: 1
                                }, $.datepicker.regional['es']
                        );

                        ctrl.$parsers.push(function(data) {
                                var date = Date.parseExact(data,
"dd/MM/yyyy");

                                ctrl.$setValidity('date', date != null);
                                if (date != null)
                                        ctrl.$setValidity('max', date <=
new Date());
                                else
                                        ctrl.$setValidity('max', true);

                                return date == null ? undefined : date;
                        });
                }
        };
});
```

En primer lugar, al usar esta directiva añadimos la funcionalidad *datepicker* al campo donde se use esta directiva. Al hacer clic sobre el campo se mostrará un calendario generado gracias al *plugin datepicker.js*, como vemos en la Figura 11.

Esta directiva no utiliza el elemento de validación el controlador *ctrl.$validators* como en la directiva *emailUca*, sino *ctrl.$parsers*. La diferencia entre *parsers* y *validators* es que los *parsers* transforman los datos de entrada antes de la validación, y si esta falla, la validación no se llevará a cabo.

En este caso, no obstante, usamos el *parser* para establecer el estado de la validación del campo *date*. Haciendo uso de un objeto fecha gracias al *plugis date.js* que *parsea* una cadena con un formato indicado en un objeto fecha comparamos que el valor de esa fecha no supera el de la fecha actual.

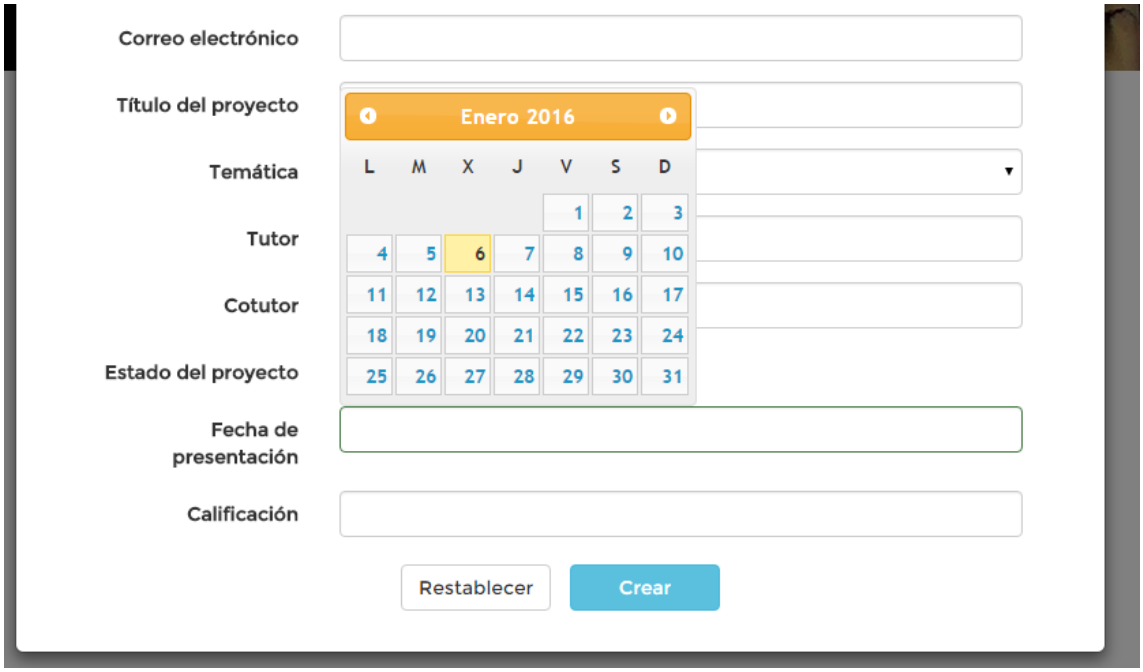

Figura 11. Calendario de *datepicker.js*

Los errores que se muestran pueden ser que la fecha introducida no tenga un formato adecuado (*create.date.$error.date*), como podemos ver en la Figura 12, o el error personalizado que indica que la fecha no puede ser superior a la actual, como vemos en la Figura 13:

```
<div ng-show="create.$submitted || create.date.$touched">
        <span class="help-block" ng-show="create.date.$error.date">Formato
de fecha inválido (dd/mm/aaaa)</span>
        <span class="help-block" ng-show="create.date.$error.max">La fecha
no puede ser mayor que la actual</span>
</div>
```

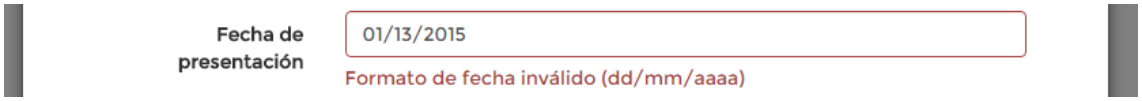

Figura 12. Fecha inválida

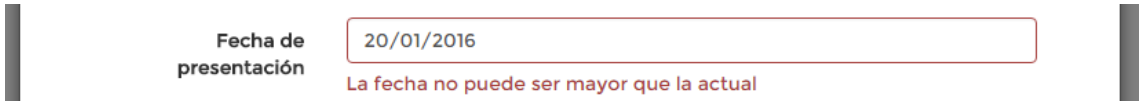

Figura 13. Fecha mayor que la actual

El último campo del formulario es la **calificación del proyecto**. Este campo, de tipo numérico (*type="number"*) validará que el campo introducido es efectivamente un número (aunque el navegador probablemente impida introducir caracteres distintos de dígitos) y, además, usando los atributos *max* y *min* definimos el rango que Angular usará para validar que la calificación introducida es un número dentro de ese rango.

```
<input type="number" ng-model="project.mark" name="mark" min="0" max="10"
class="form-control">
```

Si el número introducido es inferior al mínimo indicado, será indicado por *create.mark.$error.min*, si es superior al máximo, por *create.mark.$error.max.* Usamos estas propiedades para mostrar el mensaje de error, como podemos ver en la Figura 14:

```
<div ng-show="create.$submitted || create.mark.$touched">
        <span class="help-block" ng-show="create.mark.$error.min ||
create.mark.$error.max">La calificación debe tener un valor entre 0 y
10</span>
</div>
```

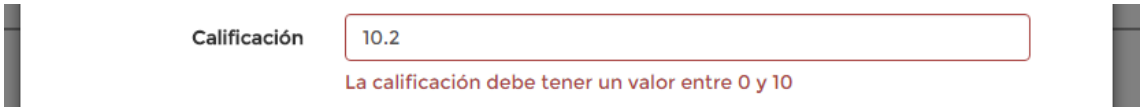

Figura 14. Calificación inválida

Con la calificación hemos visto todas las validaciones de los campos en el formulario crear, ahora veamos en la Figura 15 el aspecto de un formulario donde todos los campos son válidos y la información es correcta para su envío al servicio web y la procese.

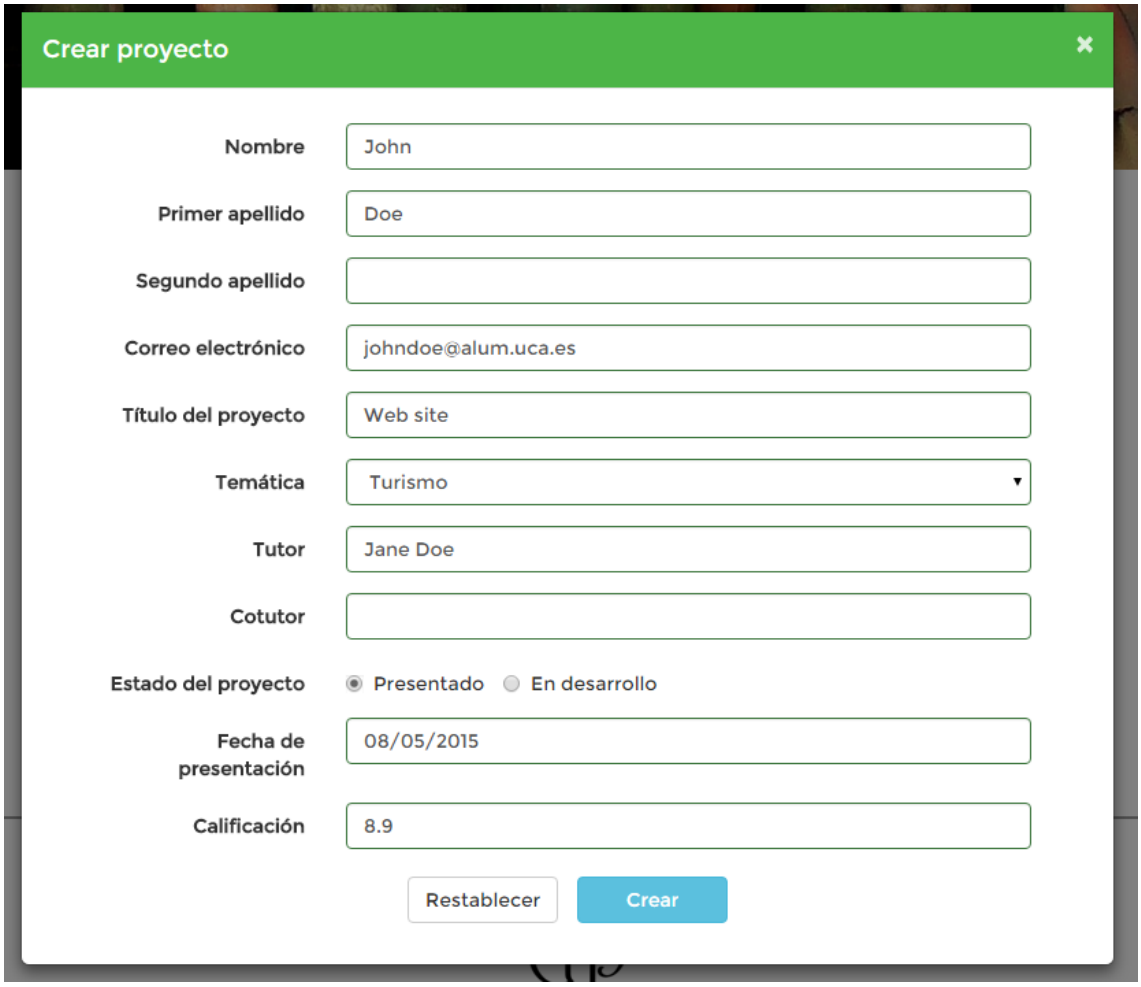

Figura 15. Formulario crear válido

Al pulsar sobre el botón "Crear" el servicio web procesará la información y creará nuestro nuevo proyecto. Recordamos que todo esto proceso es responsabilidad del método ejecutado al enviar el formulario que fue declarado en el fichero *app.js*, parte del controlador de administración, *AdminCtrl.* Al terminar con éxito la invocación del servicio tomaba la respuesta JSON y formaba un objeto cuya información formateada se mostrará en un nuevo *div* en lugar del formulario, que se ocultará (recordamos que usábamos *ng-show* y *ng-hide* en conjunción con el objeto de los resultados).

En la Figura 16 vemos el resultado de crear el nuevo proyecto:

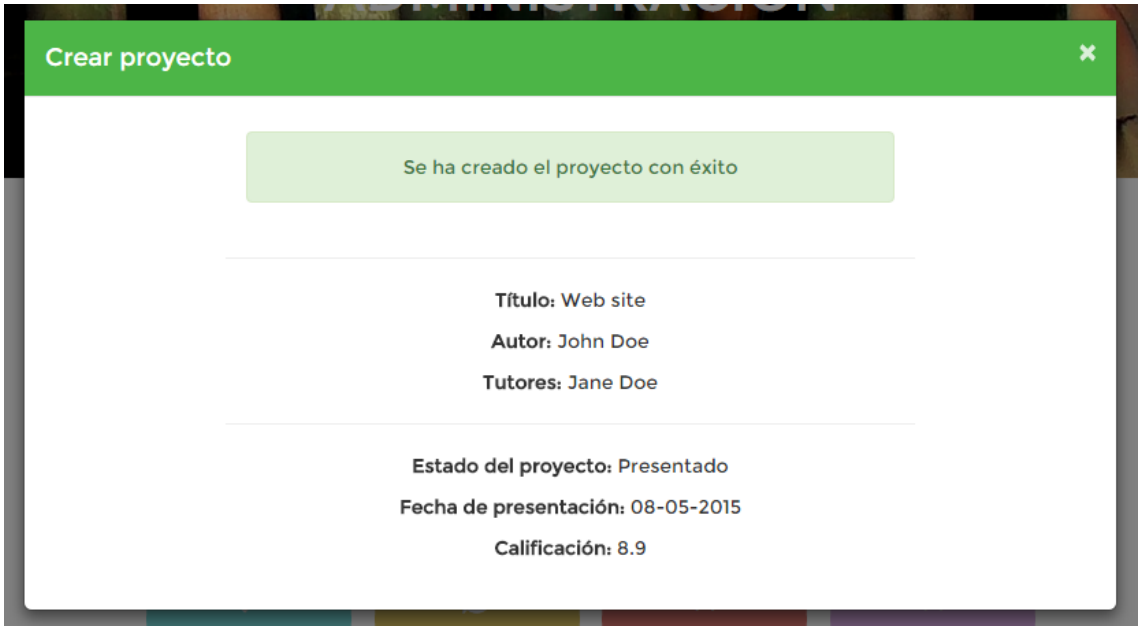

Figura 16. Resultado de crear proyecto

4.2. Formulario actualizar

Esta sección será muy breve puesto que el formulario de actualizar tiene menos campos y además es prácticamente idéntico al formulario crear, con las mismas validaciones.

En la Figura 17 podemos observar el aspecto inicial del formulario actualizar:

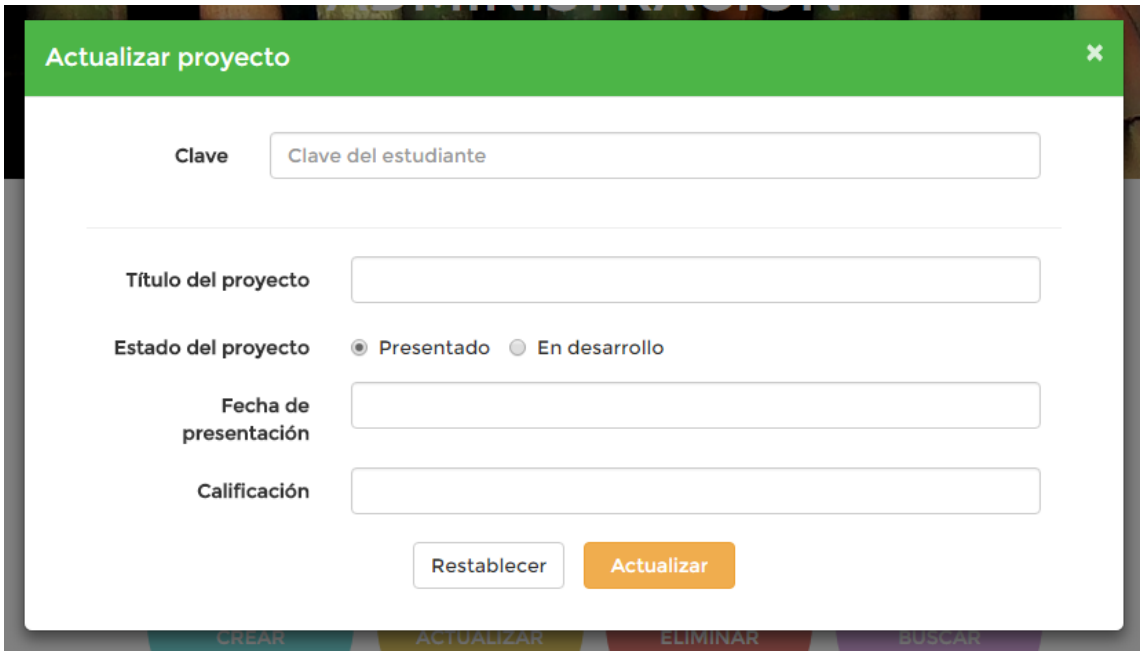

Figura 17. Formulario actualizar

Aunque la validación de este formulario es muy parecida a la del formulario crear, hay una diferencia (salvando que hay menos campos): no hay ningún campo obligatorio. Ya que existe la posibilidad de modificar únicamente un atributo del proyecto que se quiere actualizar, no podemos obligar a rellenar los demás campos.

De momento, ignoremos el primer campo "clave", y repasemos brevemente los demás campos del formulario, que contienen la información que se puede actualizar del proyecto.

El primer campo es el de **título del proyecto**, y no tiene ninguna validación, por tanto ningún mensaje de validación.

A continuación, el campo de **estado del proyecto**, que, al igual que en el formulario de crear no se valida, pero tiene la funcionalidad de *ng-change* que oculta o

muestra los campos de fecha de presentación y calificación dependiendo de qué opción se elija:

```
<input type="radio" name="state" value="presentado" ng-model="project.state"
ng-change="changeState(project.state)">
<input type="radio" name="state" value="en desarrollo" ng-
model="project.state" ng-change="changeState(project.state)" ng-
click="clearPresentedAttributes()">
```

El campo de **fecha de presentación del proyecto** aplica la misma validación que en el formulario crear. La fecha debe ser válida y se usa la directiva *date-field* para no permitir fechas superiores a la actual:

```
<input type="text" ng-model="project.date" name="date" class="form-
control" date-field>
```

Los mensajes de validación son los esperados:

```
<div ng-show="update.$submitted || update.date.$touched">
        <span class="help-block" ng-show="update.date.$error.date">Formato
de fecha inválido (dd/mm/aaaa)</span>
        <span class="help-block" ng-show="update.date.$error.max">La fecha
no puede ser mayor que la actual</span>
</div>
```

El campo de **calificación del proyecto** valida que la calificación introducida se encuentre en el rango definido por los atributos *min* y *max*, además de aplicar la validación propia de los campos numéricos:

```
<input type="number" ng-model="project.mark" name="mark" min="0" max="10"
class="form-control">
```

Y el mensaje de validación de rango:

```
<div ng-show="update.$submitted || update.mark.$touched">
        <span class="help-block" ng-show="update.mark.$error.min ||
update.mark.$error.max">La calificación debe tener un valor entre 0 y
10</span>
</div>
```

De vuelta al primer campo, el campo **clave**, que recordamos según el servidor es la clave del estudiante, definida por la concatenación de sus apellidos sin espacio entre el primero y el segundo. Este campo es usado para identificar el proyecto cuya información queremos modificar, por lo tanto es obligatorio y Angular lo valida al estar presente el atributo *required*:

```
<input required type="text" ng-model="project.key" name="key" class="form-
control" placeholder="Clave del estudiante">
```

Al usuario se le muestra un mensaje de error pertinente si la clave no está presente:

```
<div ng-show="update.$submitted || update.key.$touched">
        <span class="help-block" ng-
show="update.key.$error.required">Introduce la clave de estudiante</span>
</div>
```

En la Figura 18 podemos ver un ejemplo en el que fallen las validaciones para el formulario de actualizar:

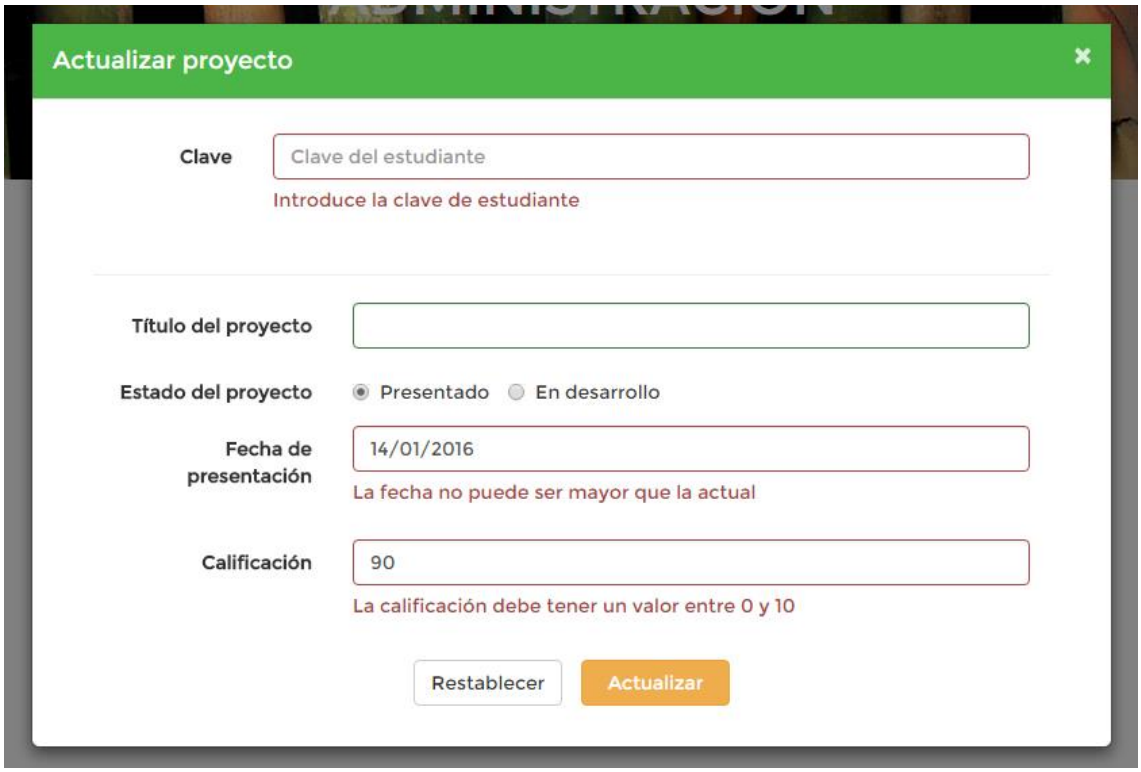

Figura 18. Formulario actualizar inválido

Intentemos ahora modificar la información del proyecto anteriormente creado, el del estudiante John Doe. Observemos en la Figura 19 el aspecto de un formulario de actualizar válido:

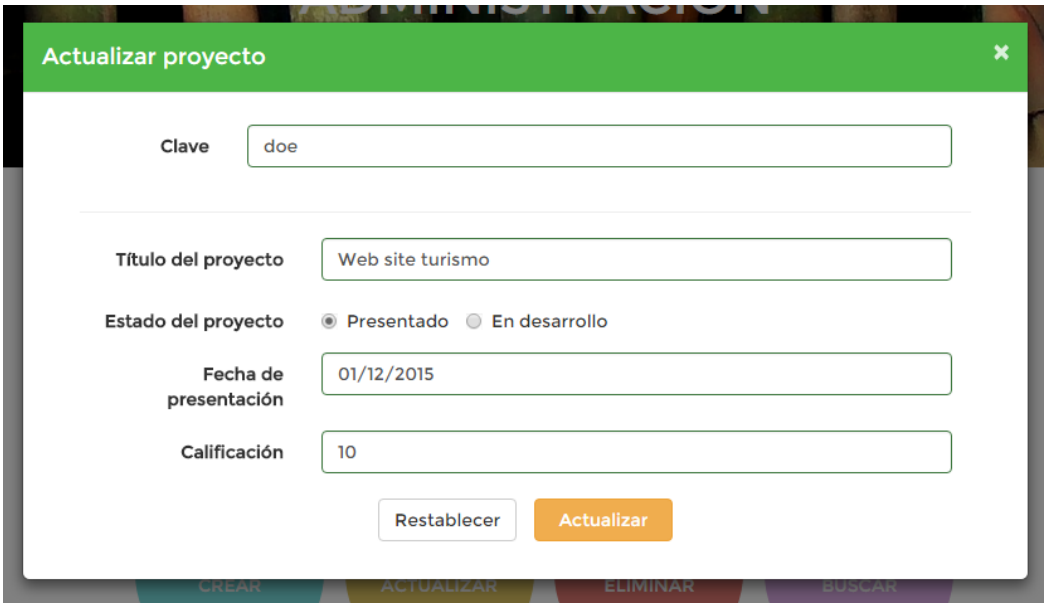

Figura 19. Formulario actualizar válido

Veamos en la Figura 20 el resultado de enviar este formulario. Se mostrará la información completa del proyecto con los datos actualizados (el servicio web devuelve un JSON con el proyecto actualizado):

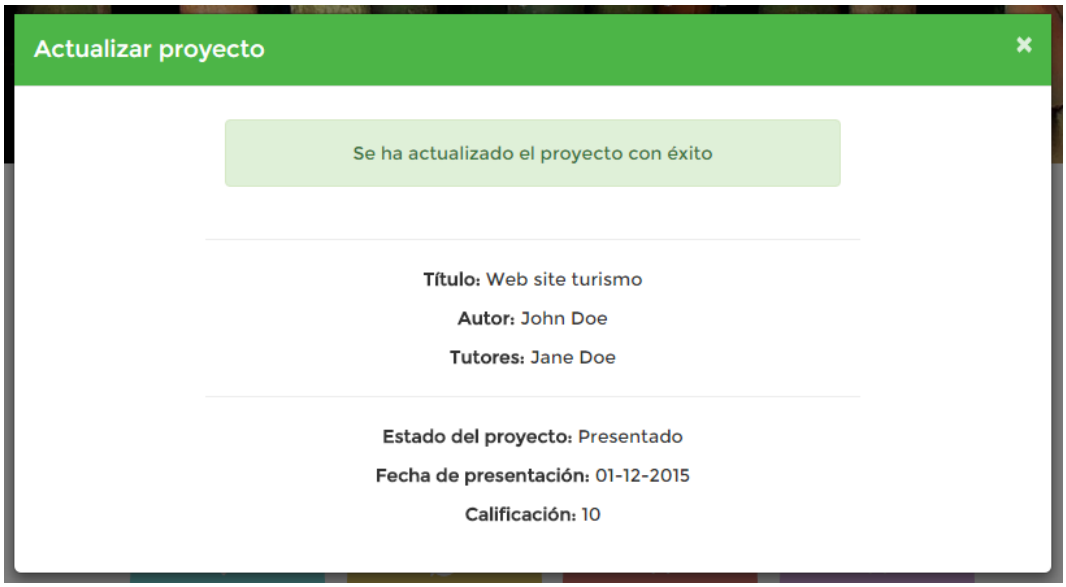

Figura 20. Resultado de actualizar proyecto

No obstante, se puede dar el caso de que el formulario sea válido pero no exista un estudiante con la clave indicada. Este caso no es responsabilidad de la validación de Angular, sino responsabilidad de la función que responde a la invocación del servicio. Recordamos que esta posibilidad estaba implementada y se mostraba un mensaje de error sobre la cabecera del *modal*¸ como se puede ver en la Figura 21:

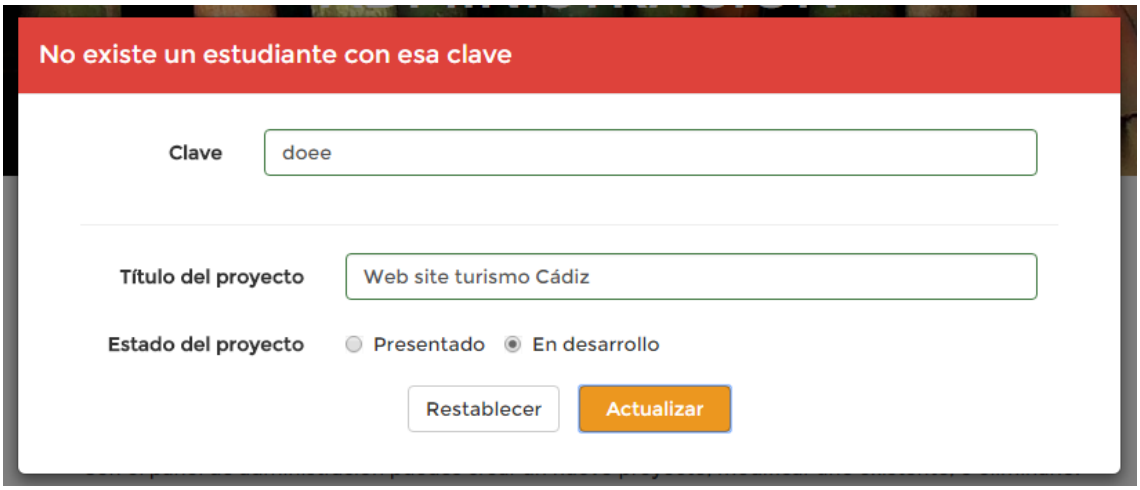

Figura 21. Formulario actualizar con estudiante inexistente

4.3. Formulario eliminar

El formulario eliminar es muy simple pues solo tiene un campo donde introducir la clave del estudiante a borrar. Podemos ver el formulario en la Figura 22.

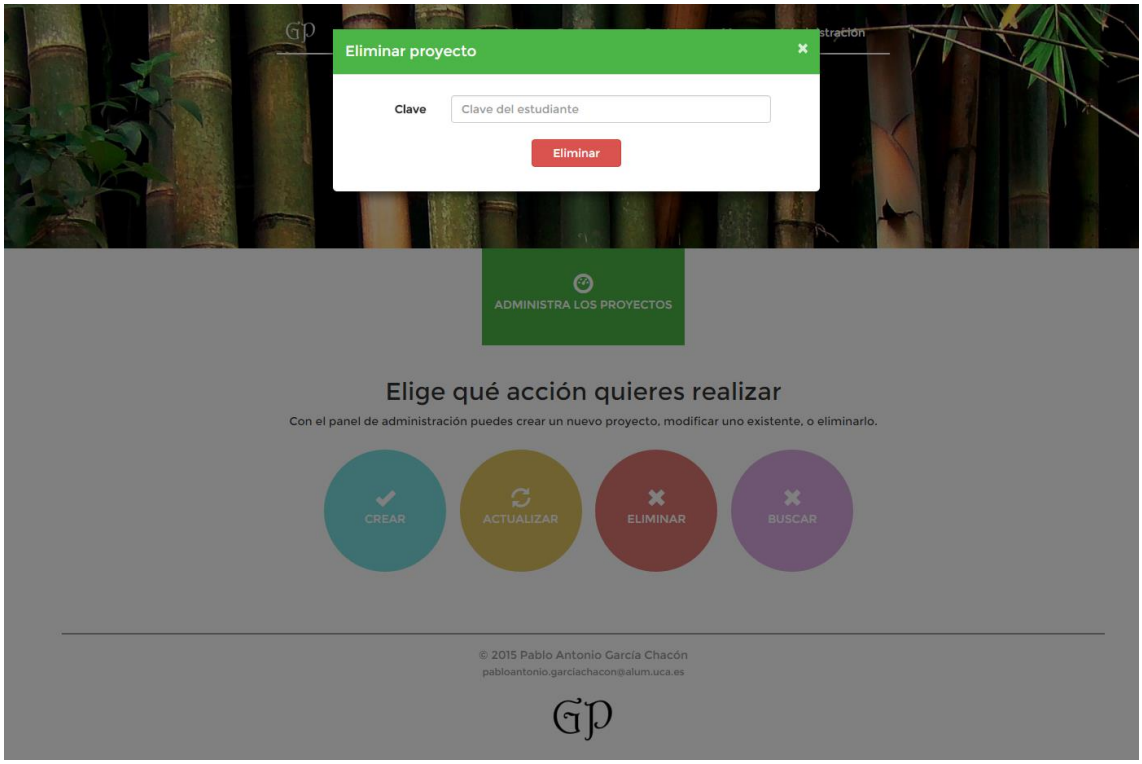

Figura 22. Formulario eliminar

Al igual que en el formulario actualizar, validamos el campo entrada de la clave, que es obligatorio:

```
<input required type="text" ng-model="key" name="key" class="form-control"
placeholder="Clave del estudiante">
```

Y mostramos un mensaje de error si no se introduce, como podemos ver en la Figura 23:

```
<div ng-show="delete.$submitted || delete.key.$touched">
        <span class="help-block" ng-
show="delete.key.$error.required">Introduce la clave de estudiante</span>
</div>
```

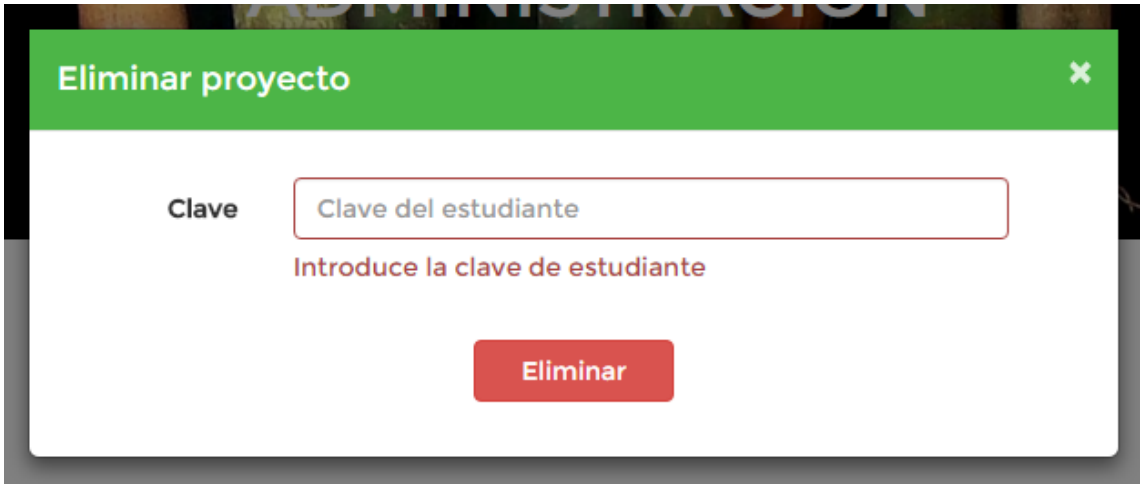

Figura 23. Formulario eliminar inválido

Un formulario válido muestra un mensaje de éxito al ser borrado el proyecto (ver Figura 25). Intentemos borrar el proyecto de John Doe, estudiante anteriormente creado (Figura 24).

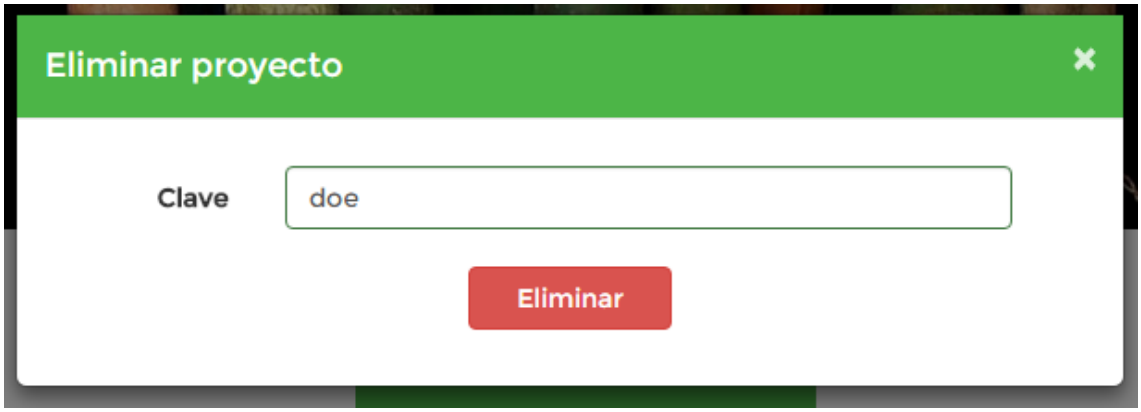

Figura 24. Formulario eliminar válido

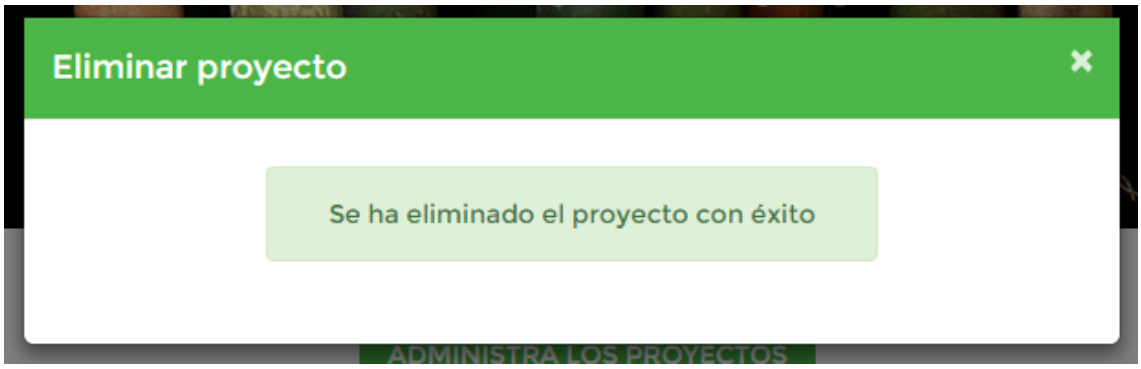

Figura 25. Resultado de eliminar proyecto

Al igual que en formulario actualizar, puede existir la posibilidad de que no se encuentre un proyecto con la clave introducida, en tal caso se mostrará un mensaje como el de la Figura 26, al intentar borrar de nuevo el proyecto del estudiante Doe, que justo acaba de ser borrado:

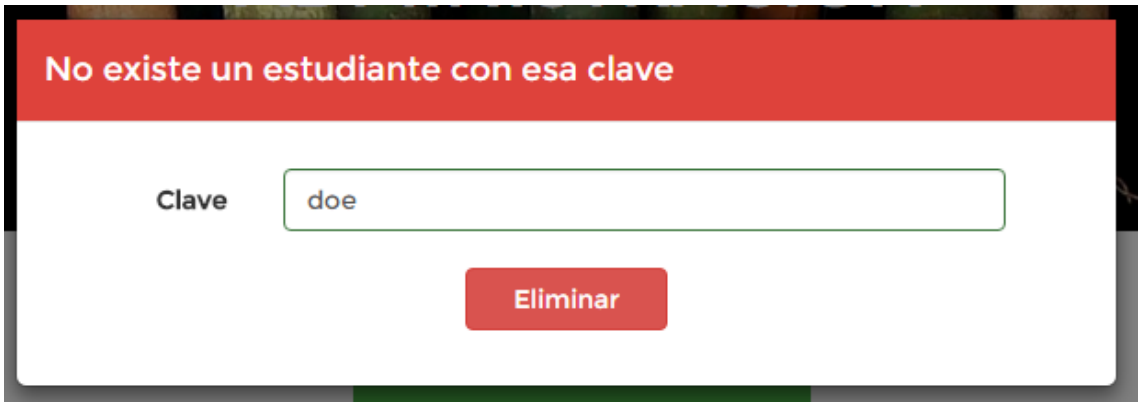

Figura 26. Formulario eliminar con estudiante inexistente

4.4. Formulario de búsqueda

El formulario de búsqueda tiene un aspecto idéntico al formulario de borrado, como se puede observar en la Figura 27. Solamente tiene un campo donde introducir la clave del estudiante, y se validará que sea obligatorio:

```
<input required type="text" ng-model="key" name="key" class="form-control"
placeholder="Clave del estudiante">
```

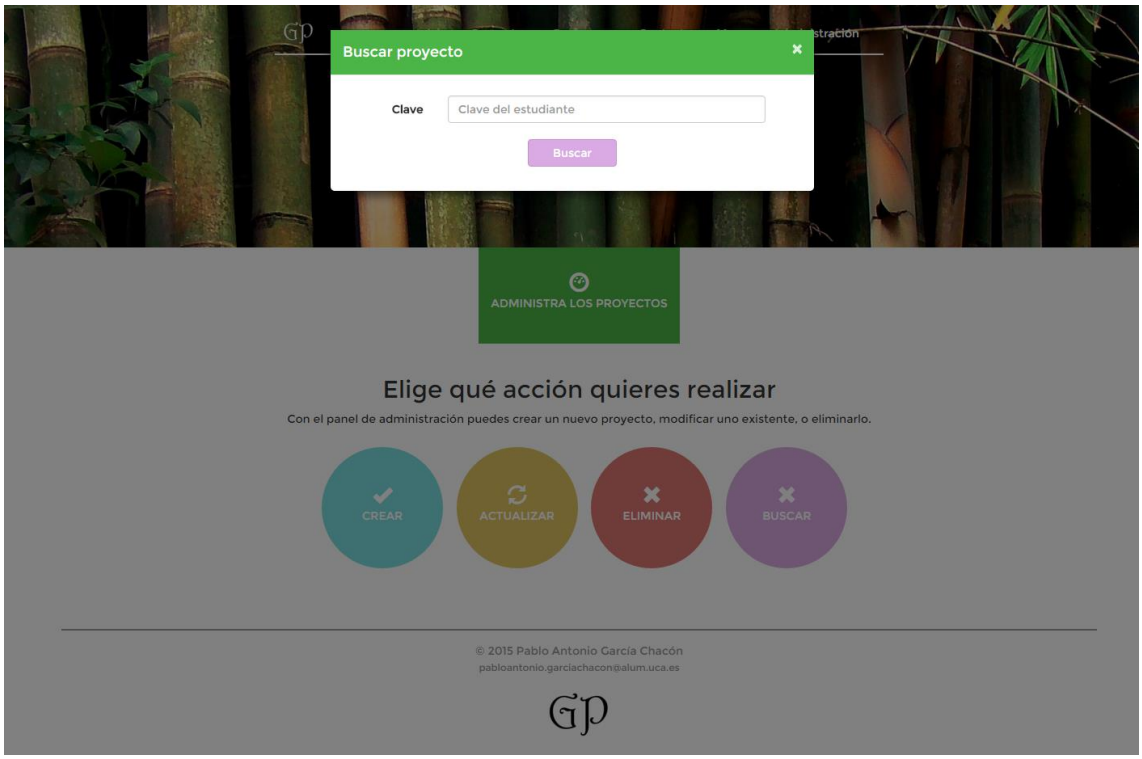

Figura 27. Formulario de búsqueda

Si no se introduce la clave se mostrará el mensaje de error esperado informando al usuario de que es obligatorio, como se ve en la Figura 28, al igual que se hacía en los formularios de actualizar y eliminar:

```
<div ng-show="search.$submitted || search.key.$touched">
        <span class="help-block" ng-
show="search.key.$error.required">Introduce la clave de estudiante</span>
</div>
```

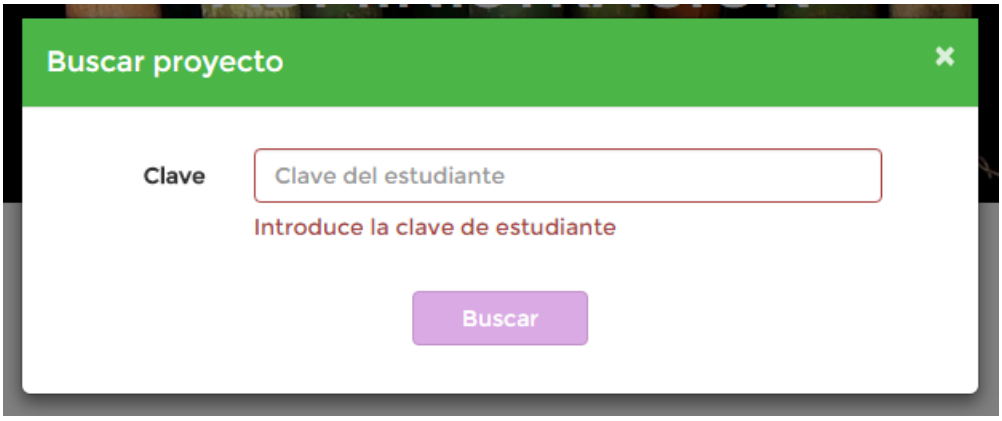

Figura 28. Formulario de búsqueda inválido

A continuación, vamos a crear de nuevo el proyecto del estudiante John Doe y vamos a buscar este proyecto preparando un formulario de búsqueda válido mostrado en la Figura 29:

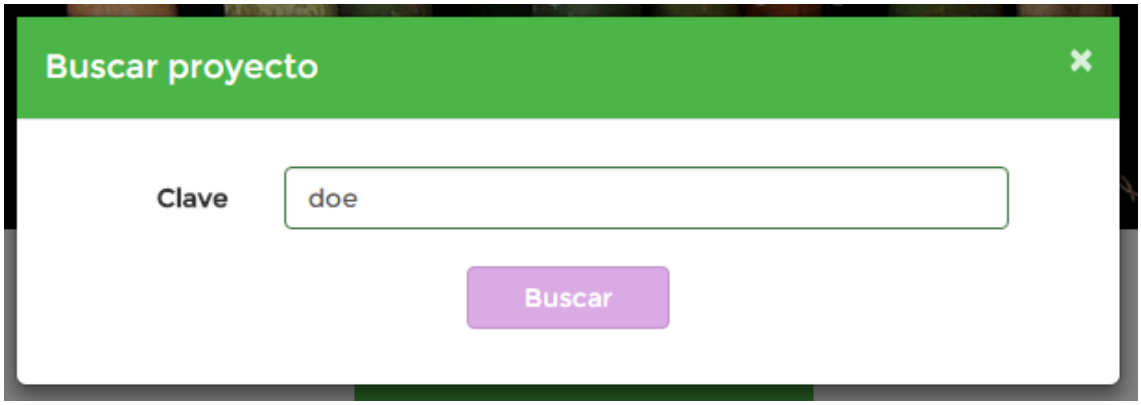

Figura 29. Formulario de búsqueda válido

En la Figura 30 se puede ver el resultado de la búsqueda:

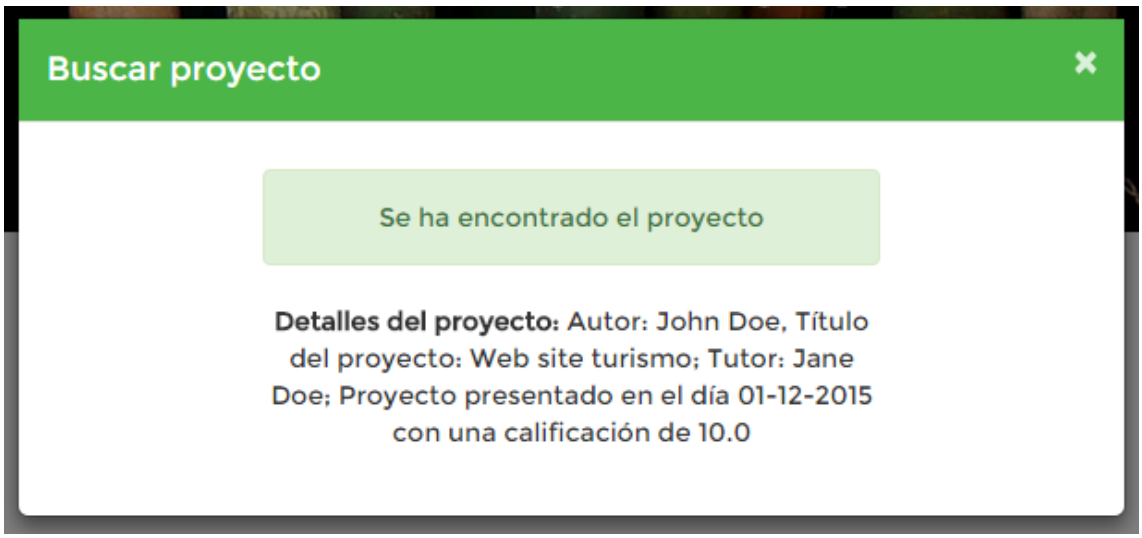

Figura 30. Resultado de buscar proyecto

Igual que con los formularios de actualizar y eliminar, si no existe un proyecto con la clave de estudiante introducida, se mostrará un mensaje de error como el de la Figura 31:

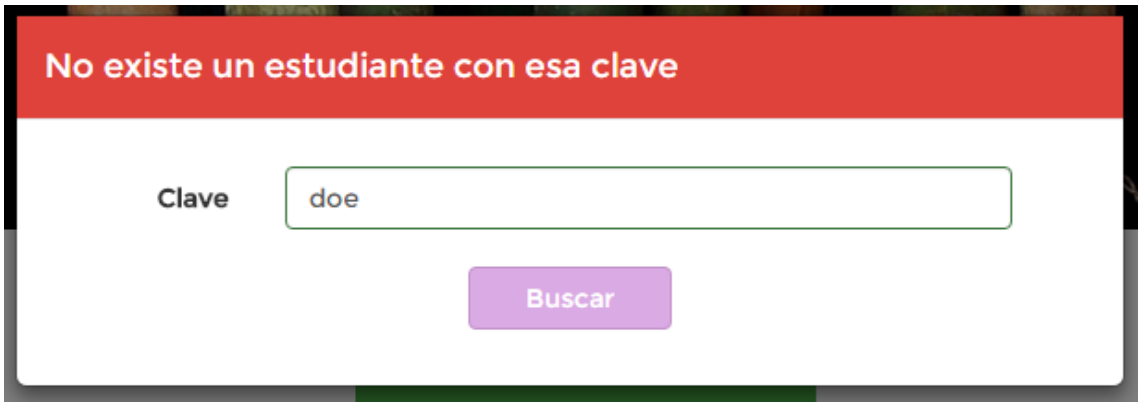

Figura 31. Formulario de búsqueda con estudiante inexistente

5. Validación de código

En esta sección evaluaremos la validez y corrección de nuestro código HTML5 y CSS3 con respecto a los estándares del W3C y que, además, cumple con los criterios de accesibilidad WAI-A WCAG 2.0.

5.1. HTML

Para validar nuestro código HTML usamos un servicio de validación del W3C. Es importante destacar que las directivas usadas por AngularJS se marcarán como errores, ya que no son atributos HTML5 estándares. Sin embargo, no hay solución posible a la hora de evitar que se evalúen estos errores porque forman parte de nuestro código original. No incluiremos dichos avisos en los resultados de validación que a continuación mostraremos.

5.1.1. Página principal

Podemos ver el resultado de la validación en la Figura 32.

Observamos tres *warnings* acerca de las cabeceras de las secciones (<*section*>). Sugiere que se deberían usar obligatoriamente cabeceras para cada sección. Sin embargo, la semántica de las secciones mencionadas por el validador, en el caso de nuestro sitio, no contempla la inclusión de cabeceras ya que se tratan de secciones generales que a su vez engloban otras secciones o elementos. Estos *warnings* aparecerán en todas las demás páginas ya que seguimos el mismo esquema de secciones y la explicación para todas ellas es el que acabamos de dar.

Por lo demás, la validación de la página es muy satisfactoria.

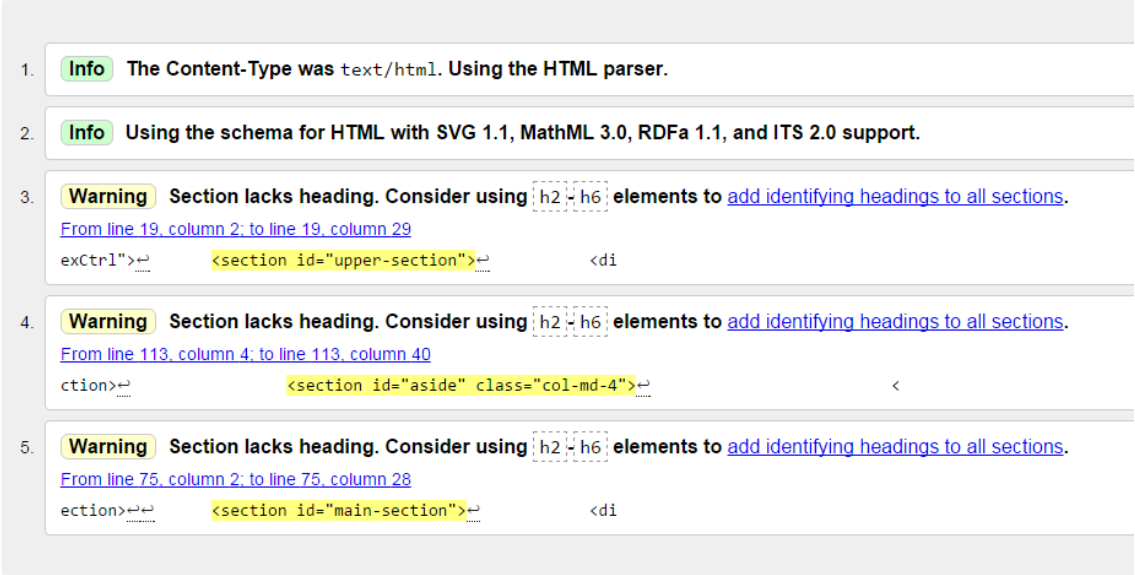

Figura 32. Validación HTML5 página principal

5.1.2. Proyectos

En la Figura 33 podemos ver la validación satisfactoria de la página de proyectos. Encontramos tres *warning* al igual que en la página inicial, y el argumento de su aparición es el mismo que el explicado anteriormente.

Hay un error que indica un valor erróneo de la imagen. En realidad no es un error porque las dobles llaves *{{}}* son usadas por AngularJS para producir una cadena. Este valor se sustituirá por el del nombre del fichero de imagen del proyecto. Es un error inevitable consecuencia del uso de Angular.

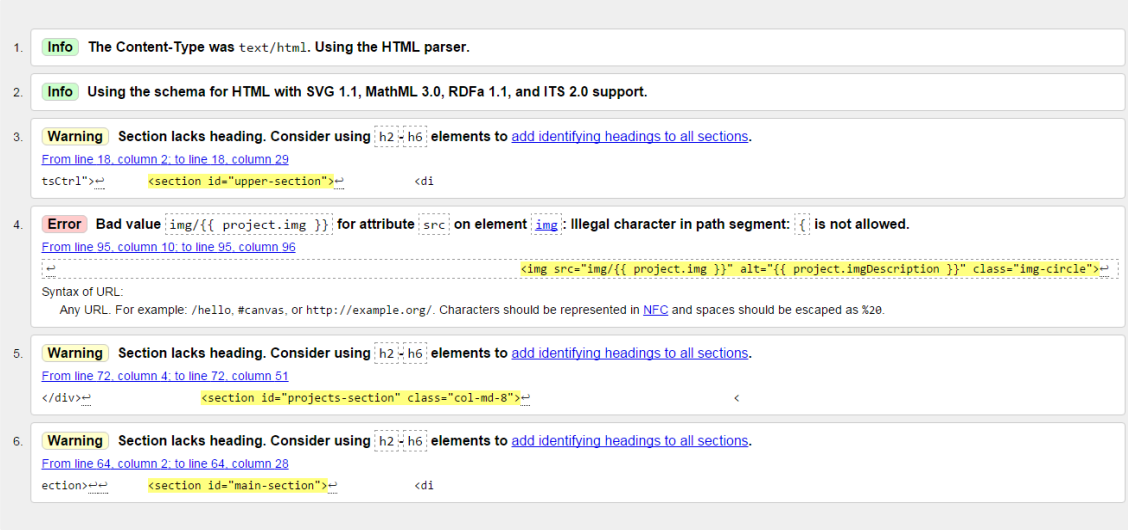

Figura 33. Validación HTML5 de la página proyectos

5.1.3. Profesores

Esta página también es validada correctamente siguiendo los estándares del W3C, como podemos ver en la Figura 34.

Tenemos tres *warnings* acerca del uso de cabeceras en ciertas secciones, al igual que vimos en las páginas anteriores y cuya explicación hemos argumentado.

Se dan tres errores en los valores de ciertos atributos del teléfono del profesor, su email y la imagen de la misma situación que en la página anterior de los proyectos en el valor de la imagen del avatar. Son valores dinámicos generados por nuestras funciones de Angular y serán sustituidos por cadenas válidas a la hora de cargar la página en el navegador.

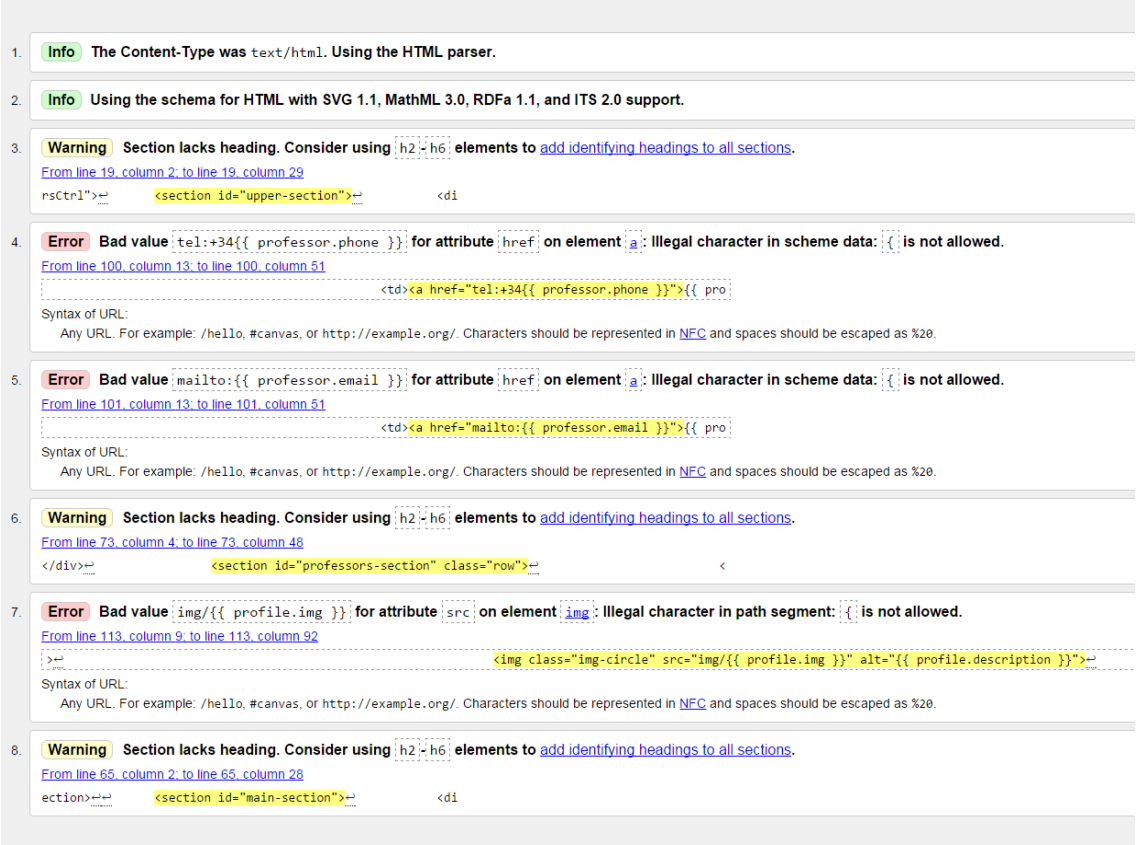

Figura 34. Validación HTML5 de la página profesores

5.1.4. Contacto

Exceptuando el usual *warning* de las cabeceras en las secciones, nuestra página de contacto usa un HTML5 válido.

Podemos ver el informe de la validación en la Figura 35:

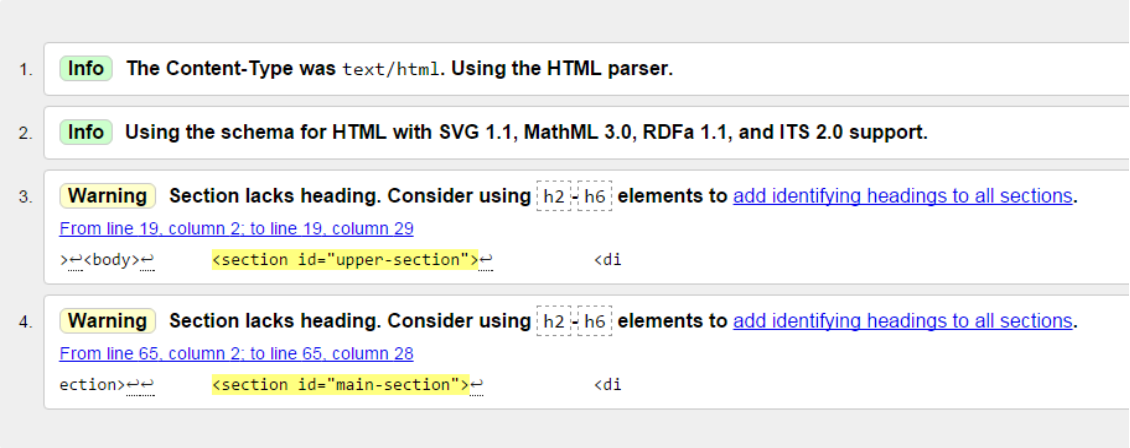

Figura 35. Validación HTML5 de la página de contacto

5.1.5. Administración

Podemos ver la validación de la página de administración en la Figura 36. No hay ningún error por lo tanto es satisfactoria.

Encontramos los *warnings* acerca de las cabeceras de las secciones mencionados anteriormente.

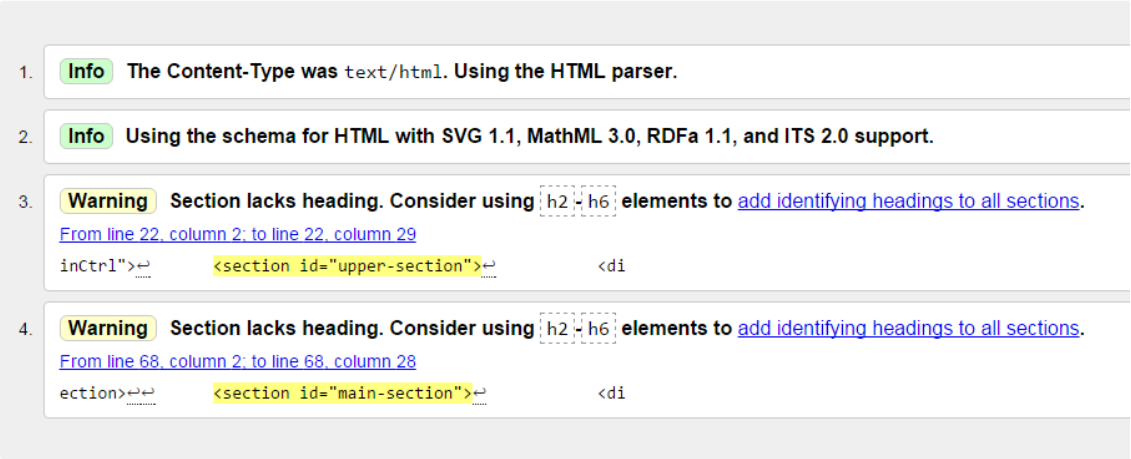

Figura 36. Validación HTML5 de la página de administración

5.2. CSS

Para la validación de nuestro CSS usamos un servicio de validación CSS3 del W3C.

No validaremos el fichero CSS de Bootstrap ya que no es nuestra responsabilidad su validez de acerudo a los estándares del W3C.

5.2.1. Estilos comunes

Validamos el fichero *app.css* con los estilos utilizados en todas las páginas del sitio web.

El servicio indica que se trata de un código CSS3 totalmente válido.

5.2.2. Estilos específicos

Al hacer uso el servicio de validación para validar los ficheros específicos de las páginas de inicio, proyectos, profesores, contacto y administración, respectivamente los ficheros *index.css, tfg.css, profesores.css, contactar.css* y *administracion.css* obtenemos una respuesta satisfactoria de este servicio: como indica la Figura 37, todo nuestro CSS es totalmente válido en su versión 3.

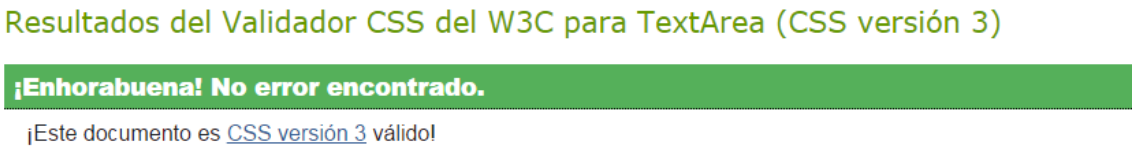

Figura 37. Validación CSS3

5.3. Accesibilidad

Para comprobar que el nivel de accesibilidad de nuestro sitio cumple con el nivel WCAG 2.0 hemos usado un servicio validador de accesibilidad de terceros.

5.3.1. Página principal

Al usar el validador no se han encontrado errores de accesibilidad en la página principal (Figura 38):

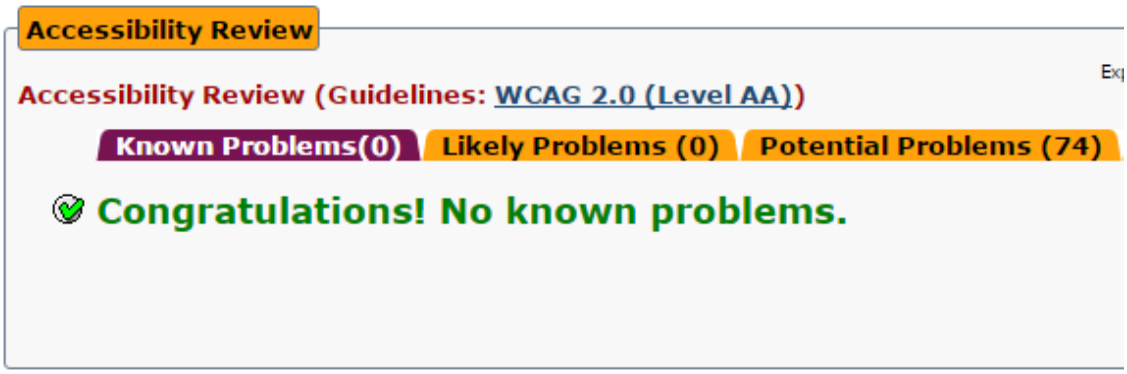

Figura 38. Accesibilidad página principal

5.3.2. Proyectos

En un principio encontramos problemas de accesibilidad en la página de proyectos, como vemos en la Figura 39, pero gracias a este servicio se han solucionado (Figura 40): se ha cambiado una etiqueta *h3* por *h2* y modificado el estilo del *h2* para que tuviese el mismo tamaño y se ha añadido un *label* al campo para filtrar por título de proyecto, con la clase de Boostrap *sr-only* que oculta el elemento a la vez que está presente para software de lectura de páginas.

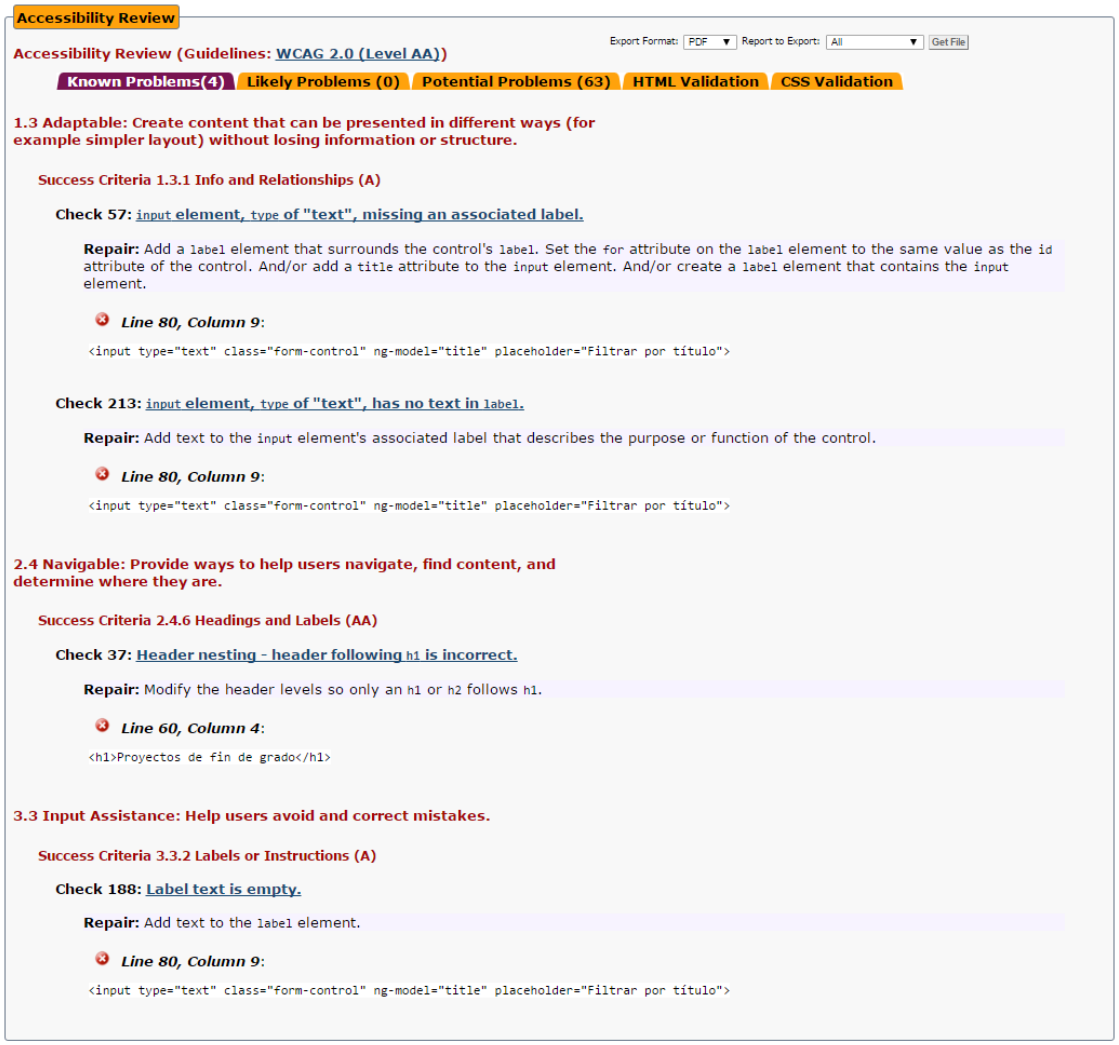

Figura 39. Problemas de accesibilidad de la página de proyectos

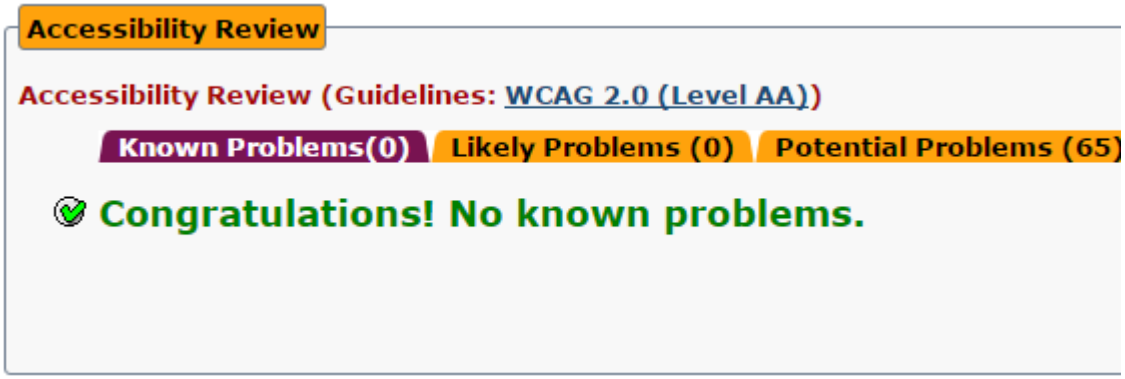

Figura 40. Accesibilidad de la página de proyectos

5.3.3. Profesores

La página de profesores es totalmente accesible como muestra la Figura 41:

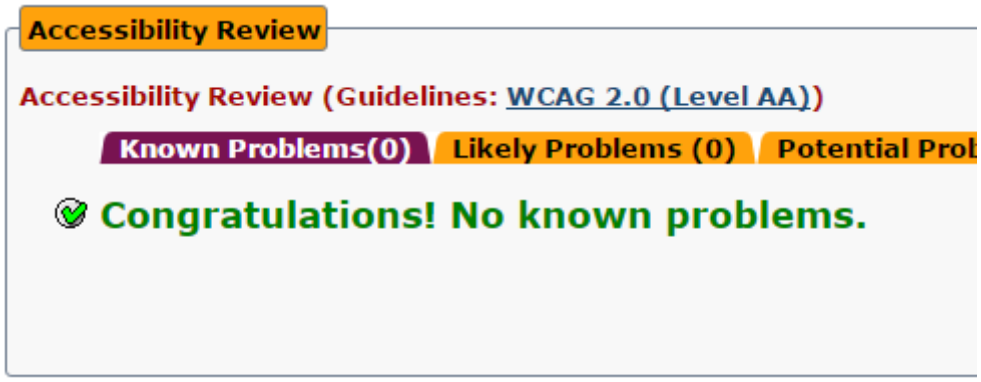

Figura 41. Accesibilidad de la página de profesores

5.3.4. Contacto

La página de contacto es también accesible cumpliendo los criterios WCAG 2.0.

5.3.5. Administración

La página de administración es también accesible cumpliendo los criterios WCAG 2.0.